Direct Input

305890
Spring 2010
3/26/2010
Kyoung Shin Park

Getting Inputs

- 기존의 Window Message 방식은 마우스나 키보드 입력을 이벤트 큐에 넣어서 처리하기 때문에 게임에 사용하기에는 입력이 느린 단점이 있다.

```
MSG msg;
ZeroMemory(&msg, sizeof(msg));
while (msg.message != WM_QUIT) {
    // check for messages
    if (PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

Getting Inputs

- GetAsyncKeyState
  - 키보드와 마우스 입력 속도를 향상시키기 위해, Window Message방식 대신 GetAsyncKeyState 함수를 사용할 수 있다.
  - 마우스의 상태나 멀티-키를 눌렸는지 확인할 수 있다.
- DirectInput
  - DirectInput은 키보드, 마우스 뿐만 아니라 조이스틱, 운전대 컨트롤 (운전 시뮬레이션 게임에 쓰이는), 게임패드 등등 컴퓨터에 연결해서 사용할 수 있는 입력장치들을 지원한다.
  - DirectInput은 다양한 입력장치를 지원하며 또한 하드웨어에 빠르고 직접적으로 접근할 수 있다.
  - DirectInput은 Window Message 방식에 의존하지 않고 윈도우 드라이버와 직접적으로 통신할 수 있다.

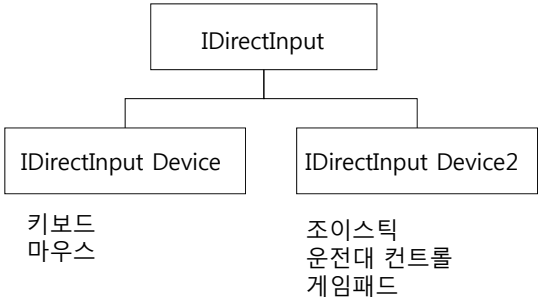
Hardware ↔ **Driver** ↔ **DirectInput** ↔ **Your Application** ↔ **User Interface**

DirectInput

- DirectInput
  - DirectInput8.0을 사용하기 위해서 d3d9.lib, d3dx9.lib, winmm.lib, dinput8.lib, dxguid.lib 을 프로젝트에 링크 & dinput.h 헤더파일을 추가
  - DirecInput에서 지원하는 입력장치
    - Keyboard – 표준 QWERTY 키보드
    - Mouse – 2-버튼 또는 3-버튼 마우스
    - Joystick – 아날로그와 디지털 조이스틱
    - 비행 Yoke – 비행 Yoke는 6 DOF (Degree-of-freedom)과 최대 32개 버튼을 지원
    - Steering Wheel Control – 아날로그/디지털 운전 시뮬레이션
    - Paddle – 다양한 회전 장치
    - Force-feedback 장치 – 기계적인 actuator가 있어서 컴퓨터가 진동시키거나 모양을 바꿀 수 있는 조이스틱 또는 다른 장치들
    - Virtual Reality Headgear Tracking 시스템 – 가상현실 사용자의 머리 위치와 회전을 지원

DirectInput Interface

- IDirectInputDevice
  - IDirectInput은 DirectInput을 사용하기 위해 생성해야 하는 기본 인터페이스이다.
  - IDirectInput8.0을 사용할 경우에는 IDirectInputDevice8 인터페이스로 키보드, 마우스, 조이스틱의 입력장치들을 사용할 수 있다.



DirectInput 설정

- DirectInput 설정
  1. DirectInput 객체를 생성한다. DirectInput8Create() 함수 사용.
  2. CreateDevice() 함수를 사용해서 하나 또는 하나 이상의 입력장치 디바이스를 생성한다.
  3. SetDataFormat() 함수를 사용해서 각각의 입력장치마다 데이터 형식을 설정한다.
  4. 각각의 입력장치 간의 협력 레벨을 설정한다. SetCooperativeLevel() 함수 사용.
  5. Acquire() 함수를 사용해서 각각의 입력장치 디바이스를 얻는다. 이 과정에서 각각의 입력장치를 DirectInputDevice8에 추가한다.
  6. GetDeviceState() 함수를 사용해서 해당 입력장치로부터 데이터를 가져온다.
  7. 마지막으로, 프로그램 종료 시에는 반드시 사용중인 객체들을 해제시킨다.

Creating DirectInput Object

- DirectInput8Create()함수를 사용하여 DirectInput객체 생성

```
HRESULT WINAPI DirectInput8Create(HINSTANCE hinst,
        DWORD dwVersion, REFIID riidltf,
        LPVOID *ppvOut, LPUNKNOWN pUnkOuter);
```

  - hInst – 해당 윈도우의 인스턴스 핸들
  - dwVersion - DirectInput의 버전. 보통 최신버전을 사용한다는 의미로 DIRECTINPUT_VERSION을 입력한다.
  - riidltf - 인터페이스 아이디. IDirectInput8.0을 사용할 것이므로 IID_IDirectInput8을 입력하면 된다.
  - ppvOut - LPDIRECTINPUT8의 객체 포인터를 대입해 준다.
  - pUnkOuter - NULL을 준다.

Creating DirectInput Object

```
HRESULT hr;
LPDIRECTINPUT8 DI_Object;

hr = DirectInput8Create(hInst, DIRECTINPUT_VERSION,
                        IID_IDirectInput8,
                                (void **) &DI_Object, NULL);
if (FAILED(hr)) return false;
```

Creating the DirectInput Device Object

- CreateDevice()함수를 사용하여 DirectInput Device객체 생성

```
HRESULT CreateDevice(REFGUID rguid,
                     LPDIRECTINPUTDEVICE *lplpDirectInputDevice,
                     LPUNKNOWN pUnkOuter);
```

  - rguid
    - 생성하려는 장치의 GUID (Globally Unique Identifier)로 Enum_Devices() 함수를 사용해서 조이스틱이나 게임패드 같은 입력장치들의 GUID를 얻을 수 있다.
    - 하지만, 키보드나 마우스의 경우에는 GUID_SysKeyboard나 GUID_SysMouse 값을 넣어주면 된다.
  - lplpDirectInputDevice – 장치 인터페이스 포인터를 넣어주면 된다. LPDIRECTINPUTDEVICE 객체의 포인터를 넘기면 된다.
  - pUnkOuter - NULL을 준다.

Creating the DirectInput Device Object

```
HRESULT hr;

// DirectInput device for Keyboard
LPDIRECTINPUTDEVICE8 pKeyboard;

// Retrieve a pointer to an IDirectInputDevice8 interface
hr = DI_Object->CreateDevice(GUID_SysKeyboard,
                             &pKeyboard, NULL);

// Check the return code from CreateDevice
if (FAILED(hr)) return false;
```

Creating the DirectInput Device Object

```
HRESULT hr;

// Create a device for the default mouse
LPDIRECTINPUTDEVICE8 pMouse;
hr = DI_Object->CreateDevice(GUID_SysMouse,
                             &pMouse, NULL);

// Check the return code from CreateDevice
if (FAILED(hr)) return false;
```

Setting the Data Format

```
HRESULT SetDataFormat(LPCDIDATAFORMAT lpdf);

typedef struct {
    DWORD dwSize;      // the size of this structure in bytes
    DWORD dwObjSize;   // the size of DIOBJECTDATAFORMAT in bytes
    DWORD dwFlags;     // Specifies attributes of this data format
    DWORD dwDataSize; // holds the size of the data packet returned
    from
                       // the input device in bytes
    DWORD dwNumObjs; // the number of objects with the rgodf array
    LPDIOBJECTDATAFORMAT rgodf; // address to an array of
                                // DIOBJECTDATAFORMAT structures
} DIDATAFORMAT, *LPDIDATAFORMAT;
```

Setting the Data Format

- 만약 사용하고자 하는 입력장치가 표준 장치가 아니면 DIDATAFORMAT 구조체를 생성하여 사용해야 한다.
- 그런데 DIDATAFORMAT 구조체를 설정은 상당히 복잡하다. 그래서 보통 미리 정의된 데이터 형식을 대입한다.
- 미리 정의된 DIDATAFORMAT 형식:
  - c_dfDIKeyboard – 키보드를 위해 정의된 데이터 형식
  - c_dfDIMouse – 마우스를 위해 정의된 데이터 형식 (4개 버튼까지 지원함)
  - c_dfDIMouse2 – 마우스나 그와 비슷한 장치를 위해 정의된 데이터 형식 (8개 버튼까지 지원함)
  - c_dfDIJoystick – 표준 조이스틱을 위해 정의된 데이터 형식
  - c_dfDIJoystick2 – 조이스틱에 확장된 기능을 지원

Setting the Data Format

```
HRESULT hr;

// set the data format for the device
// call the SetDataFormat function
hr = pKeyboard->SetDataFormat(&c_dfDIKeyboard);

// Check the SetDataFormat return code
if (FAILED(hr) return false;
```

Setting the Cooperative Level

- 협력 레벨은 사용하고자 하는 입력장치가 시스템에서 어떻게 작동할 지 알려준다.
- 입력장치를 배타적인 접근 (exclusive access) 또는 비배타적인 접근 (nonexclusive access)으로 사용할 지를 정한다.
  - Exclusive access
    - 배타적 접근으로 지정된 프로그램만이 그 특정 장치를 사용할 수 있고 다른 프로그램과 이 장치를 공유하지 않는다.
    - full-screen application으로 사용할 때 유용하다.
  - Nonexclusive access
    - 입력장치를 다른 응용프로그램과 공유한다.

Setting the Cooperative Level

- SetCooperativeLevel함수를 사용하여 협력레벨을 지정

HRESULT SetCooperativeLevel(HWND hWnd, DWORD dwFlags);

- hWnd – 윈도우 핸들
- dwFlags – 접근방법에 대한 플래그
  - DISCL_BACKGROUND – 응용프로그램이 활성화 중이거나 비활성화 중이거나 모두 입력장치들을 사용할 수 있다.
  - DISCL_FOREGROUND – 응용프로그램이 활성화 중일 때만 입력장치들을 사용할 수 있다.
  - DISCL_EXCLUSIVE – 입력장치에 대한 배타적인 접근 (exclusive access)을 하게 된다. 다른 응용프로그램에서는 해당 입력장치에 접근할 수가 없게 된다.
  - DISCL_NONEXCLUSIVE – 입력장치에 대한 비배타적인 접근 (non-exclusive access)을 하게 된다.
  - DISCL_NOWINKEY – DirectInput에게 Windows key사용하지 못하게 한다.
- 보통, DISCL_BACKGROUND | DISCL_NONEXCLUSIVE와 같이 설정해 준다.

Setting the Cooperative Level

```
// set the cooperative level
hr = pKeyboard->SetCooperativeLevel( wndHandle,
                                     DISCL_FOREGROUND |
                                     DISCL_NONEXCLUSIVE);
if (FAILED(hr)) return false;
```

Acquiring Access to the Device

- 입력 장치를 획득(Acquire) 한다.
  - 입력 장치에 대한 제어권을 얻어오고, 데이터를 받아 올 수 있도록 만드는 것이다.
  - 프로그램 종료 전에 Unacquire() 함수를 사용해서 입력장치에 대한 제어권을 꼭 반환해야 한다.

```
HRESULT Acquire(VOID);

hr = pKeyboard->Acquire();
if (FAILED(hr)) return false;
```

Enumerating Input Devices

- 대부분의 컴퓨터는 비표준 장치를 가지고 있지 않기 때문에 DirectInput에서도 비표준장치의 지원을 가정하지 않는다.
- DirectInput은 모든 사용 가능한 입력 장치를 시스템에 문의해 그것들이 접속되고 있는 지를 판정해, 그러한 장치에 대한 정보를 돌려줄 수 있다.
- 표준 키보드, 마우스를 사용하고 있는 경우는 사용 가능한 입력장치를 열거할 필요가 없다.
- 다른 모든 입력 장치 및 복수의 키보드 또는 마우스를 사용하는 시스템에 관해서는 EnumDevices() 또는 EnumDevicesBySemantics() 함수를 사용하여 사용 가능한 장치를 열거해 적절한 장치를 선택할 수 있다.

Enumerating Input Devices

```
HRESULT EnumDevices(DWORD dwDevType,
                    LPDIENUMDEVICESCALLBACK lpCallback,
                    LPVOID pvRef, DWORD dwFlags);
```

- dwDevType – 열거하는 장치의 타입을 지정. 장치 찾기를 위한 필터
  - DI8DEVCLASS_ALL – 시스템에 설치된 모든 입력장치의 리스트를 돌려준다.
  - DI8DEVCLASS_DEVICE – 다른 부류의 장치에 속하지 않는 장치 리스트를 돌려준다.
  - DI8DEVCLASS_GAMECTRL – 모든 게임 콘트롤 장치 (게임 패드나 조이스틱 같은)를 찾는다.
  - DI8DEVCLASS_KEYBOARD – 모든 키보드 장치를 찾는다.
  - DI8DEVCLASS_POINTER – 모든 포인터 장치를 찾는다.
  - 타입에 관계없이 모든 장치를 열거하는 경우는 NULL로 한다.
- lpCallback – 시스템에 있는 입력장치를 찾으려고 할 때 사용되는 사용자 지정 답신함수 (callback function). 임의의 이름으로 선언할 수 있지만, 플레이스 홀더명의 DeviceEnumCallback를 사용한다.

Enumerating Input Devices

- pvRef – 답신 함수에 건네주는 임의의 32-bit 값이며, 없으면 NULL로 지정한다.
- dwFlags
  - EnumDevice 에게 어떤 범위 (scope)에서 찾아야 할 지 알려주는 플래그
  - 예를 들어, 시스템에 설치된 장치만 찾으려 할 때, 또는 force feedback장치를 찾으려 할 때 사용한다.
  - DIEDFL_ALLDEVICES –시스템의 모든 장치. 디폴트 (default) 임.
  - DIEDFL_ATTACHEDONLY – 현재 시스템에 붙어있는 장치들만
  - DIEDFL_FORCEFEEDBACK – force feedback을 지원하는 장치들만
  - DIEDFL_INCLUDEALIASES – 윈도우는 장치에 대한 aliases를 만들어 줄 수 있다. 한 시스템의 입력장치에 대한 aliases는 다른 시스템에서는 다른 입력장치를 의미할 수 있다.
  - DIEDFL_INCLUDEHIDDEN – 감춰진 장치 (hidden devices)
  - DIEDFL_INCLUDEPHANTOMS – 단일 하드웨어가 복수의 장치로 기능 가능한 경우

Enumerating Input Devices

```
HRESULT hr;
hr = DI_Object->EnumDevices(DI8DEVCLASS_GAMECTRL,
                            DeviceEnumCallback, NULL,
                            DIEDFL_ATTACHEDONLY);
if (FAILED(hr)) return false;
```

Enumeration Devices Callback Function

- EnumDevices에 대한 호출 결과적으로 DirectInput 장치를 받는 응용프로그램 정의의 답신(callback) 함수

```
BOOL CALLBACK DeviceEnumCallback (
        LPCDIDEVICEINSTANCE lpddi, LPVOID pvRef);
```

- lpddi – 장치 인스턴스를 기술하는 DIDEVICEINSTANCE 구조체의 주소. 이 정보는 사용자에게 장치를 선택할 수 있게 출력할 때 유용하다.
- pvRef – EnumDevices 또는 EnumDevicesBySemantics에 pvRef 인수로써 건네 받는 사용자 프로그램에서 정의된 값이다.
- Returns - TRUE 또는 FALSE 대신 다음을 사용해야 한다.
  - DIENUM_CONTINUE – 열거를 속행 (continue the enumeration)
  - DIENUM_STOP – 열거를 정지 (stop the enumeration)

Enumeration Devices Callback Function

```
typedef struct {
        DWORD dwSize;
        GUID guidInstance;
        GUID guidProduct;
        DWORD dwDevType;
        TCHAR tszInstanceName[MAX_PATH];
        TCHAR tszProductName[MAX_PATH];
        GUID guidFFDriver;
        WORD wUsagePage;
        WORD wUsage;
} DIDEVICEINSTANCE, *LPDIDEVICEINSTANCE;
```

Enumeration Devices Callback Function

- dwSize - 이 구조체의 크기 (in bytes)
- guidInstance - 특정 장치를 위한 GUID
- guidProduct - 장치 ID
- dwDevType - 장치 타입
- tszInstanceName
  - Friendly name for the device, such as Joystick 1 or AxisPad
- tszProductName
  - Full product name for this device
- guidFFDriver
  - If this device supports force feedback, this value represents the GUID of the driver being used
- wUsagePage
  - Holds the Human Interface Device (HID) usage page code
- wUsage
  - Usage code for an HID

Enumeration Devices Callback Function

```
BOOL CALLBACK DeviceEnumCallback(
        const DIDEVICEINSTANCE *pdidInstance,
        VOID *pContext) {

    HRESULT hr;
    hr = DI_Object->CreateDevice(pdidInstance->guidInstance,  &pJoystick,
    NULL);

    if (FAILED(hr)) return DIENUM_CONTINUE;
    return DIENUM_STOP;
}
```

Getting the Device Capabilities

- EnumObjects 함수를 사용하여 장치로부터 특정한 구체적 정보를 얻는다.

```
HRESULT EnumObjects(
        LPDIENUMDEVICEOBJECTSCALLBACK lpCallback,
        LPVOID pvRef, DWORD dwFlags);
```

- lpCallback – 답신함수 이름
- pvRef – 답신함수에 추가적으로 보내야 할 데이터
- dwFlags – 열거하려는 입력 장치의 타입을 지정

EnumObjects Callback Function

- EnumObjects에 대한 호출 결과적으로 DirectInputDevice 개체를 받는 응용 프로그램 정의의 답신 (callback) 함수

```
BOOL CALLBACK DeviceEnumObjectCallback(
                LPCDIDEVICEOBJECTINSTANCE lpddoi,
                LPVOID pvRef);
```

- lpddoi – 열거대상의 개체를 나타내는 DIDEVICEOBJECTINSTANCE 구조체
- pvRef – EnumObjects에 pvRef 인수로서 건네 받는 응용 프로그램의 정의의 값
- Return – 열거를 속행하는 DIENUM_CONTINUE를 돌려주는지, 열거를 정지하는 DIENUM_STOP를 돌려준다.

Getting the Device State

- GetDeviceState()함수를 사용하여 해당 입력장치로부터 데이터를 가져온다.

HRESULT GetDeviceState(WORD cbData, LPVOID lpvData);

- cbData – 받는 데이터의 크기. 키보드일 경우에는 256 bytes, 마우스일 경우에는 sizeof(DIMOUSESTATE), 조이스틱일 경우에는 sizeof(DIJOYSTATE)처럼 대입해주면 된다.
- lpvData – 데이터가 저장될 곳의 포인터.

Getting the Device State

```
char KeyState[256];

if (lpdiKey->GetDeviceState(256, (LPVOID) KeyState) !=
    DL_OK)
    return false;
```

Getting Input from a Keyboard

- GetAsyncKeyState()함수를 사용하는 것처럼, 각각의 키에 대한 상수 (가상 키코드)로 입력을 받을 수 있다.
- DirectInput의 가상 키코드는 모두 DIK_로 시작된다.
  - DIK_ESCAPE – ESC 키
  - DIK_0 ~ 9 – 숫자키 0 ~ 9
  - DIK_NUMPAD0 ~ 9 – 키보드 오른쪽의 숫자 키패드 0 ~ 9
  - DIK_A ~ Z – A ~ Z 문자 키
  - DIK_RETURN – Enter 키
  - DIK_LCONTROL – 왼쪽 CTRL 키
  - DIK_RCONTROL – 오른쪽 CTRL 키
  - DIK_SPACE – 스페이스 바 키
  - DIK_F1 ~ F12 – F1 ~ F12 키
  - DIK_UP/DOWN/LEFT/RIGHT – 위/아래/왼쪽/오른쪽 화살표 키
  - DIK_TAB – Tab 키
  - DIK_PRIOR/NEXT – Page Up/Down 키

Getting Input from a Keyboard

```
// 키를 눌렀다가 떼었을 때도 계속해서 눌린 것으로 인식되는 경우가 생길 수
// 있기때문에 0x80비트를 체크. 눌림은 비트 0x80은 1, 뗌이면 비트 0x80은 0
#define KEYDOWN(name, key) (name[key] & 0x80)
char buffer[256];
while (1) {                    // main game loop
    // check the keyboard and see if any keys are currently being pressed
    g_lpDIDevice->GetDeviceState(sizeof(buffer), (LPVOID) &buffer);
    // Do seomthing with the input
    // Here KEYDOWN macro checks if the left arrow key was pressed
    if (KEYDOWN(buffer, DIK_LEFT)) {
        // Do something with the left arrow
    }
    if (KEYDOWN(buffer, DIK_UP)) {
        // Do something with the up arrow
    } }
```

Getting Input from a Mouse

- 마우스 장치를 읽는 방법은 키보드 장치를 읽는 것과 비슷하다.
- 차이점은 CreateDevice 함수에 GUID 인자와 마우스 장치에 관한 DIDATAFORMAT 구조체를 갖는다.

```
hr = DI_Object->CreateDevice(GUID_SysMouse, &pMouse, NULL);
if (FAILED(hr)) return false;
hr = pMouse->SetDataFormat(&c_dfDIMouse);
if (FAILED(hr)) return false;
```

Getting Input from a Mouse

- 마우스는 DIMOUSESTATE 타입의 버퍼가 필요하다.
- 일반적인 윈도우 프로그래밍에서 마우스를 절대좌표로 표현하는데 반해, DirectInput에서는 마우스 커서가 이전의 좌표에서 현재 좌표까지 얼마나 이동했는지의 이동량을 의미한다.

```
typedef struct DIMOUSESTATE {
    LONG lX;    // the distance the mouse has traveled in X direction
    LONG lY;    // the distance the mouse has traveled in Y direction
    LONG lZ;    // the distance the mouse has traveled in Z direction
    BYTE rgbButtons[4];  // the current state of the mouse buttons
} DIMOUSESTATE, *LPDIMOUSESTATE;
```

- 마우스 버튼 상태 체크:

```
#define BUTTONDOWN(name, key)  (name.rgbButtons[key] & 0x80)
```

Getting Input from a Mouse

```
#define BUTTONDOWN(name, key) (name.rgbButtons[key] & 0x80)
DIMOUSESTATE          mouseState;
LONG currentXpos = 320, currentYpos = 240;
while (1) {
    g_lpDIDevice->GetDeviceState(sizeof(mouseState),
                                 (LPVOID) &mouseState);
    // Do something with the input
    if (BUTTONDOWN(mouseState, 0))  {
        // Do something with the first mouse button
    }
    if (BUTTONDOWN(mouseState, 1)) {
        // Do something with the secnd mouse button
    }
    // next, check the movement of the mouse
    currentXpos += mouseState.lX;
    currentYpos += mouseState.lY;
    // do something with mouse movement
}
```

Cleaning Up DirectInput

- 프로그램 종료 전에 Unacquire 함수를 사용하여 입력 장치에 대한 제어권을 반환해야 한다.
- 그리고, 사용한 디바이스객체와 DirectInput 객체를 해제한다.

```
HRESULT Unacquire(VOID);
if (DI_Object) {
    if (DI_Device) {
        DI_Device->Unacquire();
        DI_Device->Release();
        DI_Device = NULL;
    }
    DI_Object->Release();
    DI_Object = NULL;
}
```