

# Tutorial 3: Using The Mouse

by Victor Saar

## Content

This tutorial covers the usage of the mouse with the help of DirectInput. I will only explain the important parts of the code especially on how to init the mouse and get data from it. There are some more functions, which you can use to load a new cursor or get the mouse position, etc. Something to people, who use the Microsoft IntelliPoint software. You have to set all mouse buttons to their default values. DirectInput wont be able to use them if you defined other values.

Now we come to the important things. The code is nearly the same as in the keyboard tutorial. There are mostly only some other parameters. It's the same with the mouse data. We will receive unbuffered data, like we did with the keyboard.

## cinput.h

At the beginning of this file we have some constants, which define the eight supported mouse buttons. The following vertex format and the structure are used to display the cursor. I will not explain this code here, because it's enough to fill a new tutorial and it is more Direct3D than DirectInput.

```
//constants for mouse buttons (NEW)
#define DIMOUSE_LEFTBUTTON 0
#define DIMOUSE_RIGHTBUTTON 1
#define DIMOUSE_MIDDLEBUTTON 2
#define DIMOUSE_4BUTTON 3
#define DIMOUSE_5BUTTON 4
#define DIMOUSE_6BUTTON 5
#define DIMOUSE_7BUTTON 6
#define DIMOUSE_8BUTTON 7

//vertex format for cursor (NEW)
#define D3DFVF_CURSORVERTEX (D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_TEX1)

struct D3DCURSORVERTEX
{
float fX,
    fY,
    fZ,
    fRHW;
DWORD dwColor;
float fU,
    fV;
};
```

Here we come to our DirectInput class. The most things are clear. The important new things are the function to init the mouse, the second device for the mouse and the DIMOUSESTATE2 structure, which holds the information of the current mouse state.

```
class CInput
{
public:
    CInput(DWORD, DWORD);
    ~CInput(void);
    bool InitDirectInput(void); //init DirectInput
    bool InitKeyboard(void); //init keyboard
    bool InitMouse(void); //init mouse (NEW)
    bool Update(void); //update device state
```

```

bool                KeyPressed(int);           //check for key

//draws mouse cursor (NEW)
void                DrawCursor(void);
//set new cursor (NEW)
void                SetCursor(const char*,int,int,DWORD);
//set cursor attributes (NEW)
void                SetAttributes(bool,float);
//set new cursor position (NEW)
void                SetCursorPosition(float,float);

//get relative movement (NEW)
float               GetRelativeX(void);
float               GetRelativeY(void);
//get wheel movement (NEW)
float               GetRelativeZ(void);
//get cursor position (NEW)
float               GetAbsoluteX(void);
float               GetAbsoluteY(void);
//check for mouse button (NEW)
bool                MouseButtonDown(int);

private:
LPDIRECTINPUT8      m_pDIObject;              //DirectInput object
LPDIRECTINPUTDEVICE8 m_pDIKeyboardDevice;     //keyboard device
LPDIRECTINPUTDEVICE8 m_pDIMouseDevice;       //mouse device (NEW)

char                m_KeyBuffer[256];        //buffer for keys

LPDIRECT3DTEXTURE9 m_pCursorTexture;        //cursor texture (NEW)
//mouse state structure (NEW)
DIMOUSESTATE2      m_MouseState;
bool                m_bInverted;             //inverted y axis (NEW)
float               m_fSensitivity,         //mouse sensitivity (NEW)
                  m_fCursorX,              //cursor position (NEW)
                  m_fCursorY;

int                 m_iHotSpotX,            //cursor hot spot (NEW)
                  m_iHotSpotY;

DWORD               m_dwAxes,               //number of axes (NEW)
                  m_dwButtons,             //number of buttons (NEW)
                  m_dwCursorColor,         //cursor color (NEW)
                  m_dwScreenWidth,         //screen width (NEW)
                  m_dwScreenHeight;        //screen height (NEW)
};

```

## cinput.cpp

Now we can start with the functions. First we have to add `InitMouse()` to the constructor. When you create the object you have to pass the screen dimensions to it. They are needed for the cursor.

```

if(!InitDirectInput()) g_App.SetD3DStatus(false);
else if(!InitKeyboard() || !InitMouse()) g_App.SetD3DStatus(false);

```

And, of course, we also have to clean-up at the end. That's why we also have to change the destructor.

```

if(m_pDIMouseDevice != NULL)
{
m_pDIMouseDevice->Unacquire();
m_pDIMouseDevice->Release();
m_pDIMouseDevice = NULL;
}

```

Now comes the most important stuff. Before we can use the mouse we have to init it, of course. This is done like with the keyboard. At the beginning we declare a structure, which we will use later. Then we create the device with the help of the DirectInput object.

```
bool CInput::InitMouse(void)
{
    DIDEVCAPS MouseCapabilities; //device capabilities

    if(FAILED(m_pDIObject->CreateDevice(GUID_SysMouse,
                                       &m_pDIMouseDevice,
                                       NULL)))
    {
        MessageBox(g_App.GetWindowHandle(),
                  "CreateDevice() failed!",
                  "InitMouse()",
                  MB_OK);
        return false;
    }
}
```

The next thing is to set the data format. We set it to `c_dfDIMouse2`, which declares that we want to use the system mouse with up to eight buttons.

```
if(FAILED(m_pDIMouseDevice->SetDataFormat(&c_dfDIMouse2)))
{
    MessageBox(g_App.GetWindowHandle(),
              "SetDataFormat() failed!",
              "InitMouse()",
              MB_OK);
    return false;
}
```

Now we have to set the cooperative level. We set it to `DISCL_BACKGROUND` and `DISCL_NONEXCLUSIVE`. For more information on this take a look in the DirectX SDK documentation.

```
if(FAILED(m_pDIMouseDevice->SetCooperativeLevel(g_App.GetWindowHandle(),
                                               DISCL_BACKGROUND |
                                               DISCL_NONEXCLUSIVE)))
{
    MessageBox(g_App.GetWindowHandle(),
              "SetCooperativeLevel() failed!",
              "InitMouse()",
              MB_OK);
    return false;
}
```

Before we can get data from the mouse we have to acquire it.

```
if(FAILED(m_pDIMouseDevice->Acquire()))
{
    MessageBox(g_App.GetWindowHandle(),
              "Acquire() failed!",
              "InitMouse()",
              MB_OK);
    return false;
}
```

At the end we use the structure from the beginning. The next part of code fills the structure with the device capabilities and checks if the mouse is connected with the computer. So we can also find out if we have a mouse wheel or how many buttons the mouse has.

```

MouseCapabilities.dwSize = sizeof(MouseCapabilities);
m_pDIMouseDevice->GetCapabilities(&MouseCapabilities);

if(!(MouseCapabilities.dwFlags & DIDC_ATTACHED))
{
    MessageBox(g_App.GetWindowHandle(),
        "Mouse is currently not attached!",
        "InitMouse()",
        MB_OK);
    return false;
}

m_dwAxes = MouseCapabilities.dwAxes;
m_dwButtons = MouseCapabilities.dwButtons;

SetCursor("arrow.bmp", 0, 0, 0xffffffff);
SetAttributes(false, 1.0f);

return true;
} //InitMouse

```

Now we want to get data from the mouse. We only have to add a small part of code to the `Update()` function, which updates the current device states. We fill the `DIMOUSESTATE2` structure with the mouse state. If the function returns `DIERR_INPUTLOST` we acquire the device again, because it has been lost.

```

if(DIERR_INPUTLOST == m_pDIMouseDevice->GetDeviceState(sizeof(m_MouseState),
    (LPVOID)&m_MouseState))
{
    m_pDIMouseDevice->Acquire();
}

```

The last function I will explain is `MouseButtonDown()`. The parameter is the mouse button, which we want to check. You can use the constants from the beginning to check the buttons. If the high-order bit of the array is set the button is down and we return `true`.

```

bool CInput::MouseButtonDown(int Button)
{
    if(m_MouseState.rgbButtons[Button] & 0x80)
    {
        return true;
    }

    return false;
} //MouseButtonDown

```

That's all I will explain, because I think you can find out the rest by yourself. If you have questions to the not explained code let me know.