Contact

Contact me here:
Twitter: @dastopher
Email: chris@directxtutorial.com

Or leave me feedback:
Quick Feedback

Previous Next

Lesson Overview

Making your game fullscreen is easy, but requires changing a few details of the program, as well as adding a couple lines of code.

In this lesson we will cover two things. First, we will go over how to globalize your screen resolution and why you would do this. Second, we'll cover the mechanics of making a window go into fullscreen mode and back again.

Setting Up the Screen Size

Throughout your DirectX experience and in game programming you will come across many functions and structs that demand to know your screen size. This can become a hassle when you decide to change the resolution later, and especially when you decide to change it during run-time. For right now, we will cover a simple method to standardize your screen size across your program.

First, we must add two directives to the top of our program. These represent the screen width and the screen height.

```
// define the screen resolution
#define SCREEN_WIDTH  800
#define SCREEN_HEIGHT 600
```

The next step is to go through your program to where you indicate the width and height of your window. Up to this point in the tutorial, you only have two, although we will come across another in a minute. Do this to the code (changes in bold):

```
    hWnd = CreateWindowEx(NULL,
                          L"WindowClass",
                          L"Our Direct3D Program",
                          WS_OVERLAPPEDWINDOW,
                          300, 300,
                          SCREEN_WIDTH, SCREEN_HEIGHT,    // set window to new resolution
                          NULL,
                          NULL,
                          hInstance,
                          NULL);
```

And this:

```
    viewport.TopLeftX = 0;
    viewport.TopLeftY = 0;
    viewport.Width = SCREEN_WIDTH;
    viewport.Height = SCREEN_HEIGHT;
```

In a later lesson we will cover how to maintain screen size throughout your game after changing it during runtime.

There are specific resolutions that are available on most PCs, the most common of which can be seen in this table.

Resolution	Pixels	Widescreen
800 x 600	480,000	No
1024 x 768	786,432	No

1152 x 864	995,328	No
1280 x 1024	1,310,720	No
1600 x 1200	1,920,000	No
1440 x 900	1,296,000	Yes
1680 x 1050	1,764,000	Yes
1920 x 1080	2,073,600	Yes
1920 x 1200	2,304,000	Yes

Changing to Fullscreen Mode

While almost all major games are played in fullscreen mode, many games include the ability to switch between fullscreen and windowed mode. While we want fullscreen by default, we also want the user to have the ability to switch easily between one and the other. This is always done using the Alt-Enter keys.

When upgrading to full screen, there are several things we need to do:

1. Modify the window to have no background.
2. Set the back buffer to a specific size.
3. Set DirectX to automatically switch when Alt-Enter is used.
4. Modify the CleanD3D() function to turn off fullscreen when closing.

1. Modify the window to have no background.

To remove the window's background, all we need to do is comment one of the members of the WINDOWCLASSEX struct that was used to set the background color. Doing this leaves the background color untouched, which means it won't be visible as a window for a second or two before the game starts (important to making your game look professional).

```
// wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
```

2. Set the back buffer to a specific size.

Next, we have to tell DirectX about our new screen resolution. We do this by making a few changes to the scd struct we built in the last lesson.

```
void initD3D(HWND hWnd)
{
    DXGI_SWAP_CHAIN_DESC scd;    // create a struct to hold various swap chain information

    ZeroMemory(&scd, sizeof(DXGI_SWAP_CHAIN_DESC));    // clear out the struct for use

    scd.BufferCount = 1;                                    // one back buffer
    scd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;     // use 32-bit color
    scd.BufferDesc.Width = SCREEN_WIDTH;                    // set the back buffer width
    scd.BufferDesc.Height = SCREEN_HEIGHT;                  // set the back buffer height
    scd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;      // how swap chain is to be used
    scd.OutputWindow = hWnd;                                // the window to be used
    scd.SampleDesc.Count = 1;                               // how many multisamples
    scd.SampleDesc.Quality = 0;                             // multisample quality level
    scd.Windowed = TRUE;                                    // windowed/full-screen mode

    // ...
```

3. Set DirectX to automatically switch when Alt-Enter is used.

This step is quite simple. All we need to do is add a flag to the scd struct. The rest is handled for us. We can put this flag in the Flags member of the DXGI_SWAP_CHAIN_DESC.

```
void initD3D(HWND hWnd)
{
    DXGI_SWAP_CHAIN_DESC scd;    // create a struct to hold various swap chain information

    ZeroMemory(&scd, sizeof(DXGI_SWAP_CHAIN_DESC));    // clear out the struct for use

    scd.BufferCount = 1;                                    // one back buffer
    scd.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;     // use 32-bit color
    scd.BufferDesc.Width = SCREEN_WIDTH;                    // set the back buffer width
    scd.BufferDesc.Height = SCREEN_HEIGHT;                  // set the back buffer height
    scd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;      // how swap chain is to be used
    scd.OutputWindow = hWnd;                                // the window to be used
    scd.SampleDesc.Count = 1;                               // how many multisamples
    scd.SampleDesc.Quality = 0;                             // multisample quality level
    scd.Windowed = TRUE;                                    // windowed/full-screen mode
    scd.Flags = DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH;     // allow full-screen switching

    // ...
```

4. Modify the CleanD3D() function to turn off fullscreen when closing.

Direct3D is actually incapable of closing when in fullscreen mode. This is due to certain threading issues that occur behind the scenes. To correctly close down, we must make sure that we are in windowed mode. We can use the function SetFullscreenState() to do this. Here's what the new CleanD3D() function will look like:

```
// this is the function that cleans up Direct3D and COM
void CleanD3D(void)
{
    swapchain->SetFullscreenState(FALSE, NULL);    // switch to windowed mode

    // close and release all existing COM objects
    swapchain->Release();
    backbuffer->Release();
    dev->Release();
    devcon->Release();
}
```

The first parameter of the function is which state you wish to switch to. FALSE indicates windowed mode, while TRUE indicates fullscreen mode.

The second parameter is a feature which could allow you to select which video adapter to use. This is useful for specific cases where you use multiple monitors. However, for almost every game, you could just set this to NULL and it will choose the correct adapter.

The Finished Program

So, there isn't much to change. Let's take a look at the whole picture and see what is different and what is the same.

[Main.cpp]

There isn't really a point in me showing a screenshot of this program's result, because it would just be a blue rectangle. Your program should look like that: a blue rectangle with a mouse pointer in it.

Ready...Set...

Great! We now have a simple app that can correctly switch to full screen and back. It's a simple upgrade, but it's necessary for an actual game.

Exercises

1. Change the resolution of the program until you are familiar with the various resolutions selectable.

When you're ready to go on, let's hit the next lesson!

Next Lesson: Drawing a Triangle

GO! GO! GO!