Contact

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

Or leave me feedback:
Quick Feedback

Lesson 4: Window Size and Client Size

Previous Next

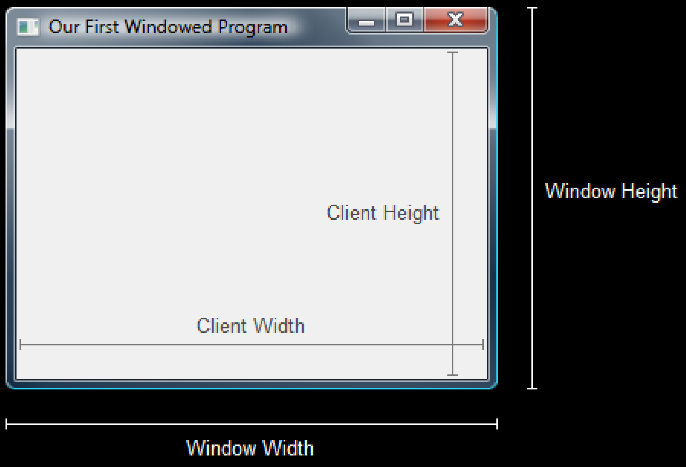
Lesson Overview

When working with graphics, it is important to know the exact size of the area in which you draw. In the last lesson, we built a window and set its size to 500 x 400. However, the area Direct3D will draw in is *not* 500 x 400 for that window.

In this lesson we are goint to discover the actual size of the drawing area and learn a function to more accurately set it.

Window Size vs. Client Size

When we called CreateWindowEx(), we used 500 and 400 to set the size of the window. However, this differs from the size of the *client*. The client area is the portion of the window that does not include its border.

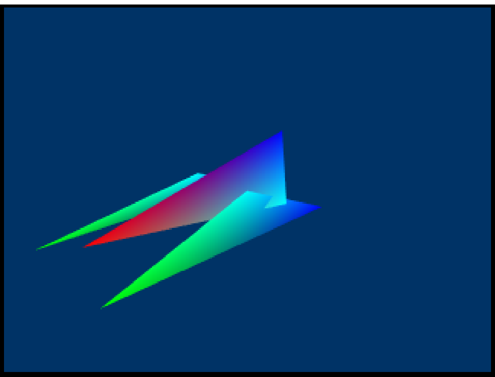
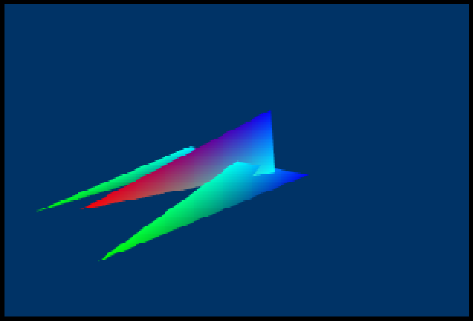


Client Size and Window Size

As you can see here, the window size extends from the edges of the border, while the client size extends the interior of the border. When rendering, we will only be drawing on the client area of the window. Therefore, it is important to know its exact size.

Why is this important exactly? When drawing using Direct3D, you are asked to specify the size of the image to be produced. If the client area of your window is a different size than this image, it gets stretched or shrunk to fit the client area.

Let's compare two screenshots of an example program in a later lesson. The image on the left was taken normally, while the one on the right was taken without AdjustWIndowRect().

Rendering With and Without AdjustWindowRect()

As you can see, the screenshot on the right has some obvious distortions. These were created when the image was shrunk to fit the client area.

The AdjustWindowRect() Function

Rather than setting the window size and then determining the client size, it is ideal to determine the client size in advance, and then calculate the appropriate window size. To do this we will use the function AdjustWindowRect() before creating the window.

This is really a simple function. What it does is take the desired size and position of our client area, and calculate the necessary window position and size to create that client size.

Here is the prototype of the function:

```
BOOL AdjustWindowRect(LPRECT lpRect,
                      DWORD dwStyle,
                      BOOL bMenu);
```

The first parameter is a pointer to a RECT struct. The RECT that is pointed to contains the coordinates of the desired client area. When the function is called, the RECT is modified to instead contain the coordinates of the window area.

The second parameter is the window style. The function uses this information to determine the size of the window border.

The third parameter is a BOOL value, telling the function whether or not we are using menus. Menus are not technically part of the client area, so it must be taken into consideration.

What does this function look like in real code? Let's take a look. The following is a modification of our call to CreateWindowEx():

```
RECT wr = {0, 0, 500, 400};    // set the size, but not the position
AdjustWindowRect(&wr, WS_OVERLAPPEDWINDOW, FALSE);    // adjust the size

// create the window and use the result as the handle
hWnd = CreateWindowEx(NULL,
                      L"WindowClass1",
                      L"Our First Windowed Program",
                      WS_OVERLAPPEDWINDOW,
                      300,    // x-position of the window
                      300,    // y-position of the window
                      wr.right - wr.left,    // width of the window
                      wr.bottom - wr.top,    // height of the window
                      NULL,
                      NULL,
                      hInstance,
                      NULL);
```

There are a few new lines of code here, shown in bold. Let's take a look at each one and find out what they do exactly.

RECT wr = {0, 0, 500, 400};

This is a simple statement. We create a rect (which I've named 'wr' for window rect), and initialize it with the size of the desired client area. We don't put position in the 'left' and 'top' values because for what we are doing we don't need them.

AdjustWindowRect(&wr, WS_OVERLAPPEDWINDOW, FALSE);

After the RECT is initialized, we call the AdjustWindowRect() function. We fill it with the address of the RECT, the window style, and FALSE to indicate there is no menu.

wr.right - wr.left,
wr.bottom - wr.top,

When AdjustWindowRect() is called, the window width will be the difference between the right and the left, and the height will be the difference between the bottom and the top.

Using these two expressions for the width and the height of the window will give us the correct size of the window.

Adding the New Code

Here is our new code, including the AdjustWindowRect() function. The parts that have changed are now in **bold**.

[Main.cpp]

When you run this program, you will find that there is actually no difference in the outcome. The truth is, you will not see the difference until you use AdjustWindowRect() in game programming.

While this was a very simple lesson, it is nevertheless an important thing to know when programming any kind of graphics software.

Alright, one more lesson to go and we're on to Direct3D!

Next Lesson: The Real-Time Message Loop

GO! GO! GO!