|  |  |  |
| --- | --- | --- |
| **K.M.S.P. Mandal’s**  **Sant Rawool Maharaj Mahavidyalay, Kudal**  **Department Of Computer Science** | Date :-  Roll No :-05  Expt No :-1 | **Signature** |
| **Title** :- Setup DirectX 11, Window Framework and Initialize Direct3D Device. | | |

Introduction To DirectX:-

Direct3D is a low-level graphics API (application programming interface) that enables us to render 3D worlds using 3D hardware acceleration. Essentially, Direct3D provides the software interfaces through which we control the graphics hardware. Having the Direct3D layer between the application and the graphics hardware means we do not have to worry about the specifics of the 3D hardware, so long as it is a Direct3D 11 capable device.

A Direct3D 11 capable graphics device must support the entire Direct3D 11 capability set, with few exceptions. This is in contrast to Direct3D 9, where a device only had to support a subset of Direct3D 9 capabilities; consequently, if a Direct3D 9 application wanted to use a certain feature, it was necessary to first check if the available hardware supported that feature, as calling a Direct3D function not implemented by the hardware resulted in failure. In Direct3D 11, device capability checking is no longer necessary because it is now a strict requirement that a Direct3D 11 device implement the entire Direct3D 11 capability set.

Direct3D 11 is a rendering library for writing high performance 3D graphics applications using modern graphics hardware on the Windows platform. (A modified version of DirectX 9 is used on the XBOX 360.) Direct3D is a low-level library in the sense that its application programming interface (API) closely models the underlying graphics hardware it controls. The predominant consumer of Direct3D is the games industry, where higher level rendering engines are built on top of Direct3D. However, other industries need high performance interactive 3D graphics as well, such as medical and scientific visualization and architectural walkthrough. In addition, with every new PC being equipped with a modern graphics card, non-3 D applications are beginning to take advantage of the GPU (graphics processing unit) to offload work to the graphics card for intensive calculations; this is known as general purpose GPU computing, and Direct3D 11 provides the compute shader API for writing general purpose GPU programs.

**Initializing Direct 3D:**

1. Create the ID3D11Device and ID3D11DeviceContext interfaces using the D3D11CreateDevice function.

2. Check 4X MSAA quality levelsupport using the ID3D11Device::CheckMultisampleQualityLevels method.

3. Describe the characteristics of the swap chain we are going to create by filling out an instance of the DXGI\_SWAP\_CHAIN\_DESC structure.

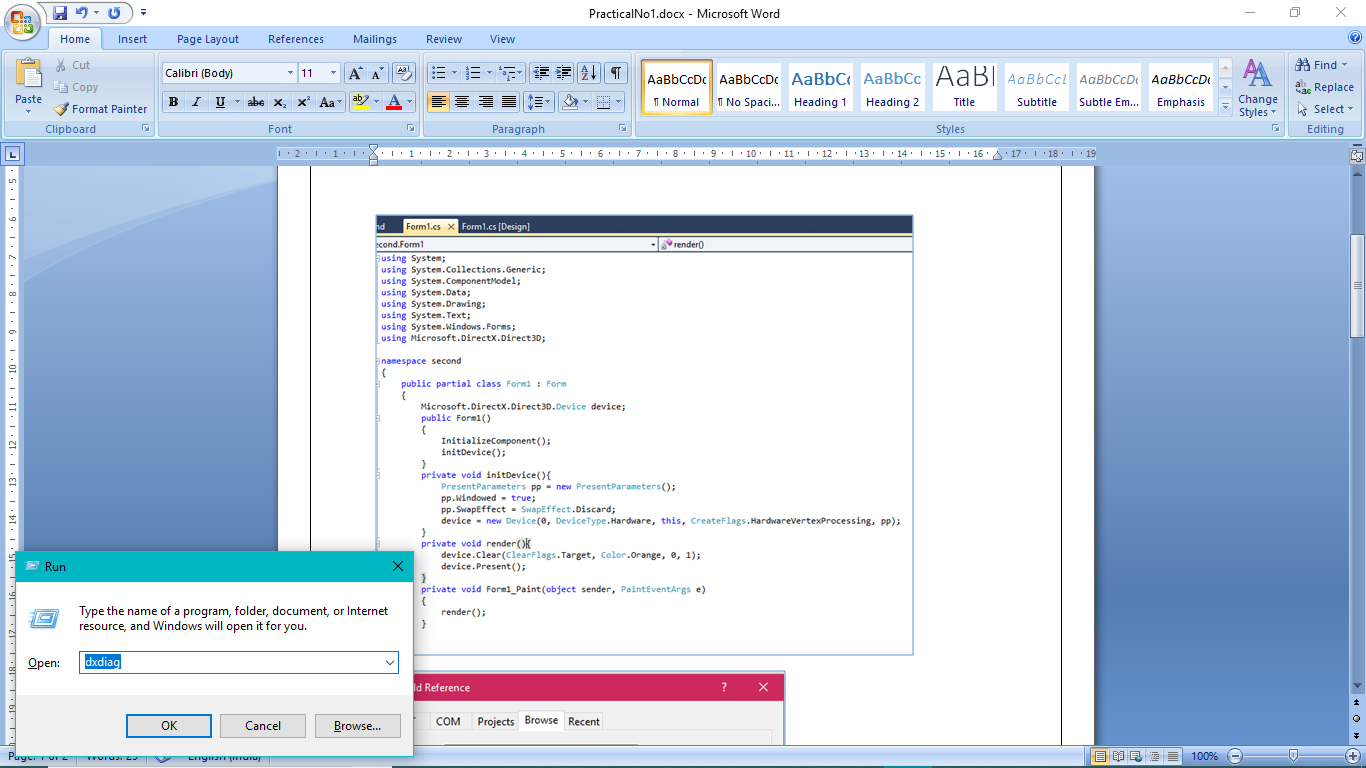
4. Query the IDXGIFactory instance that was used to create the device, and create an IDXGISwapChain instance.

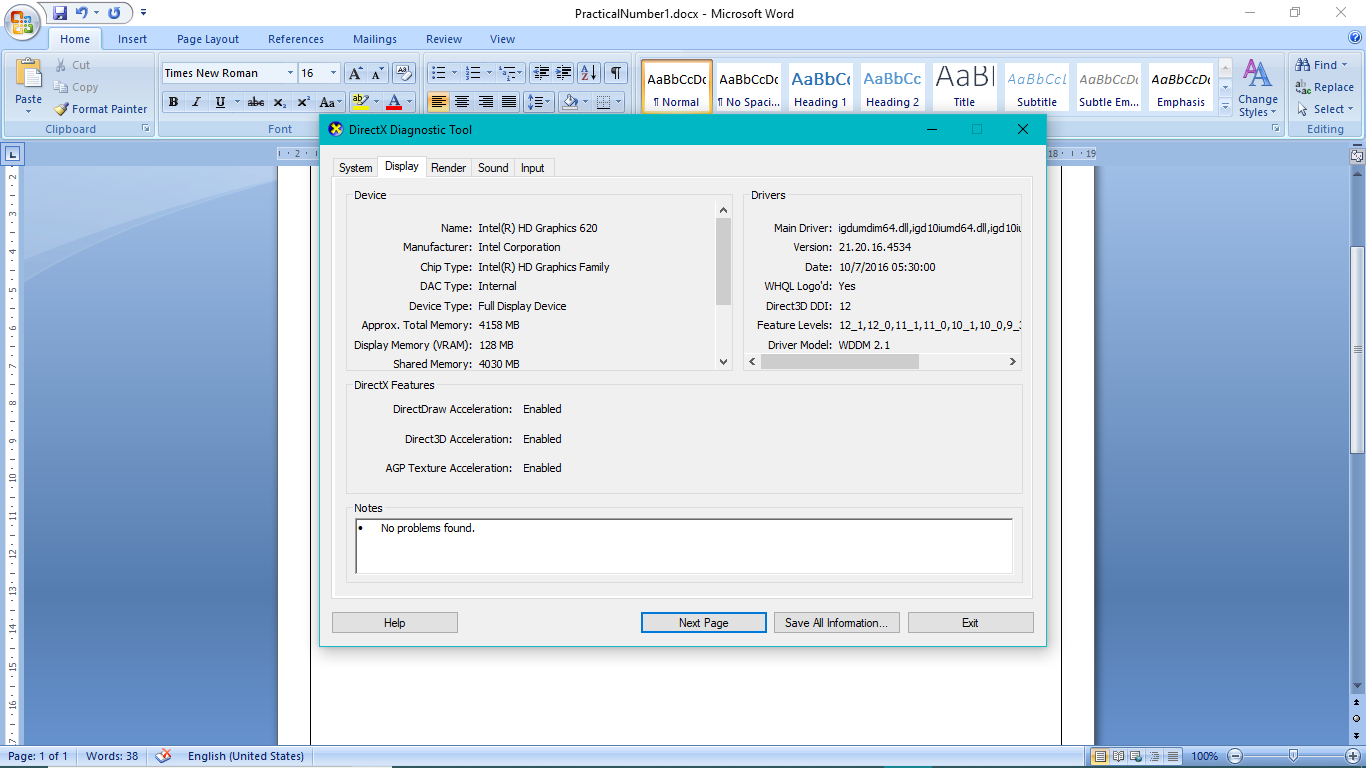
5. Create a render target view to the swap chain’s back buffer.

6. Create the depth/stencil buffer and its associated depth/stencil view.

7. Bind the render target view and depth/stencil view to the output merger stage of the rendering pipeline so that they can be used by Direct3D. 8. Set the viewport.

**Checking if acceleration is enabled or not:-**





**Window Framework and Initialize DirectX 3D Device:-**

* **Create the Device and Context:**

Initializing Direct3D begins by creating the Direct3D 11 device (ID3D11Device) and context (ID3D11DeviceContext). These two interfaces are the chief Direct3D interfaces and can be thought of as our software controller of the physical graphics device hardware; that is, through these interfaces we can interact with the hardware and instruct it to do things .

1. The **ID3D11Device interface** is used to check feature support, and allocate resources.

2. The **ID3D11DeviceContext interface** is used to set render states, bind resources to the graphics pipeline, and issue rendering commands.

**1. DriverType**: In general, you will always specify D3D\_DRIVER\_TYPE\_HARDWARE for this parameter to use 3D hardware acceleration for rendering. However, some additional options are:

**D3D\_DRIVER\_TYPE\_REFERENCE:**

Creates a so-called reference device . The reference device is a software implementation of Direct3D with the goal of correctness (it is extremely slow because it is a software implementation). The reference device is installed with the DirectX SDK and is available to developers only; it should not be used for shipping applications. There are two reasons to use the reference device:

(i) To test code your hardware does not support; for example, to test Direct3D 11 code when you do not have a Direct3D 11 capable graphics card.

(ii) To test for driver bugs. If you have code that works correctly with the reference device, but not with the hardware, then there is probably a bug in the hardware drivers.

**2. Windowed:** Specify true to run in windowed mode or false for full-screen mode.

**3. SwapEffect:** Specify DXGI\_SWAP\_EFFECT\_DISCARD in order to let the display driver select the most efficient presentation method.

**4.Flags:** Optional flags. If you specify DXGI\_SWAP\_CHAIN\_FLAG\_ALLOW\_MODE\_SWITCH, then when the application is switching to full-screen mode, it will choose a display mode that best matches the current back buffer settings. If this flag is not specified, then when the application is switching to full-screen mode, it will use the current desktop display mode.

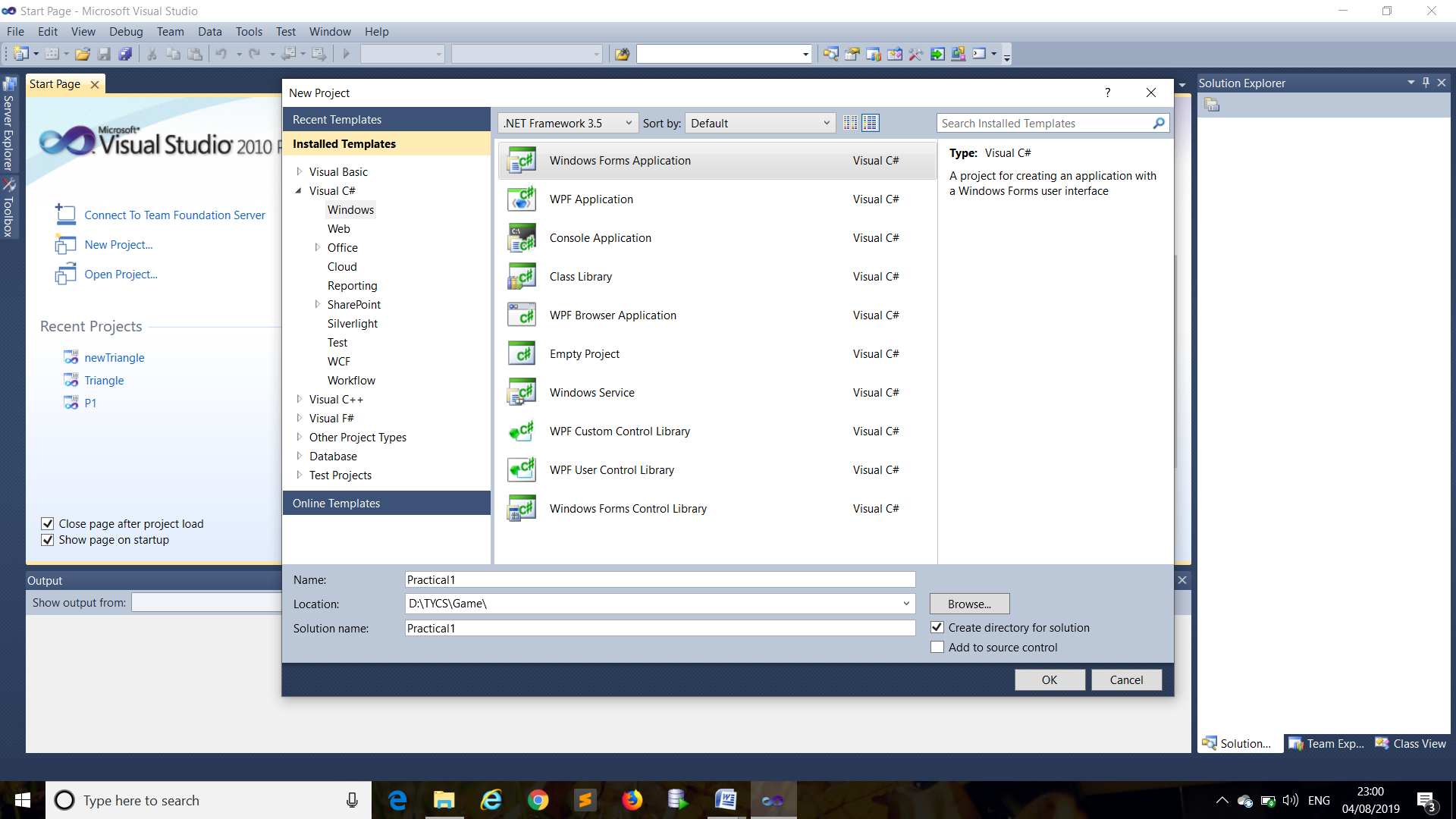
**Create the render Target View:**

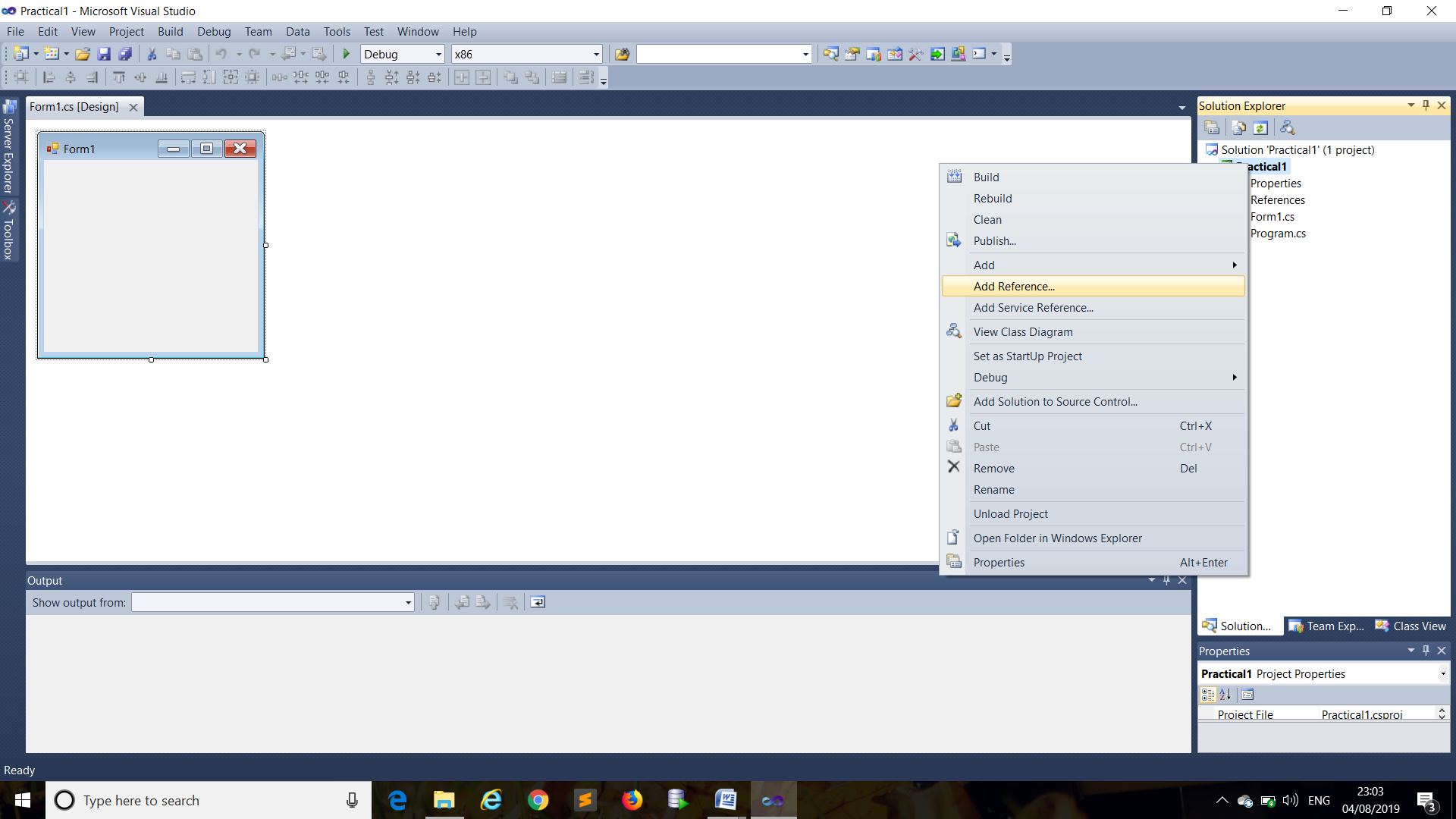
We do not bind a resource to a pipeline stage directly; instead, we must create a resource view to the resource and bind the view to the pipeline stage. In particular, in order to bind the back buffer to the output merger stage of the pipeline (so Direct3D can render onto it), we need to create a render target view to the back buffer.

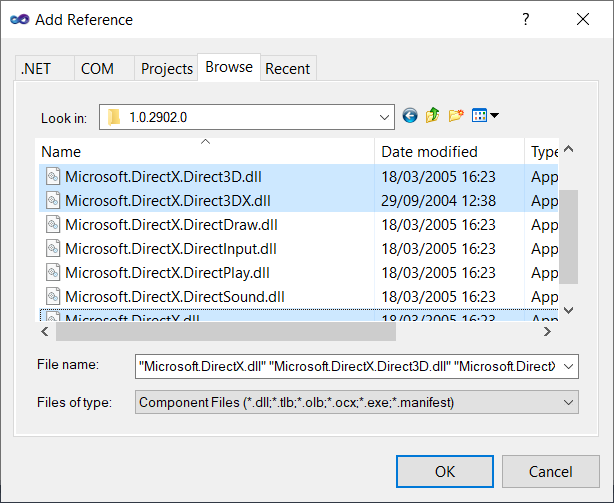
**Create the Depth/Stencil Buffer and View:**

We now need to create the depth/stencil buffer. The depth buffer is just a 2D texture that stores the depth information (and stencil information if using stenciling).

**Create Windows Forms Application**



Linking the DirectX Libraries:- 



Program:-

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using Microsoft.DirectX;

using Microsoft.DirectX.Direct3D;

namespace Practical1

{

public partial class Form1 : Form

{

Device d;

Microsoft.DirectX.Direct3D.Font font;

public Form1()

{

InitializeComponent();

Initdevice();

InitFont();

}

public void Initdevice()

{

PresentParameters p = new PresentParameters();

p.Windowed = true;

p.SwapEffect = SwapEffect.Discard;

d = new Device(0, DeviceType.Hardware, this, CreateFlags.HardwareVertexProcessing, p);

}

public void InitFont()

{

System.Drawing.Font f = new System.Drawing.Font("Comic Sans MS",20f,FontStyle.Bold);

font = new Microsoft.DirectX.Direct3D.Font(d, f);

}

private void render()

{

d.Clear(ClearFlags.Target, Color.Yellow, 0, 1);

d.BeginScene();

d.VertexFormat = CustomVertex.PositionColored.Format;

font.DrawText(null, "Game Programming", new Point(10, 10), Color.Violet);

d.EndScene();

d.Present();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

d.Clear(ClearFlags.Target, Color.DarkSlateBlue, 1.0f, 0);

d.Present();

render();

}

}

}

Output:-

