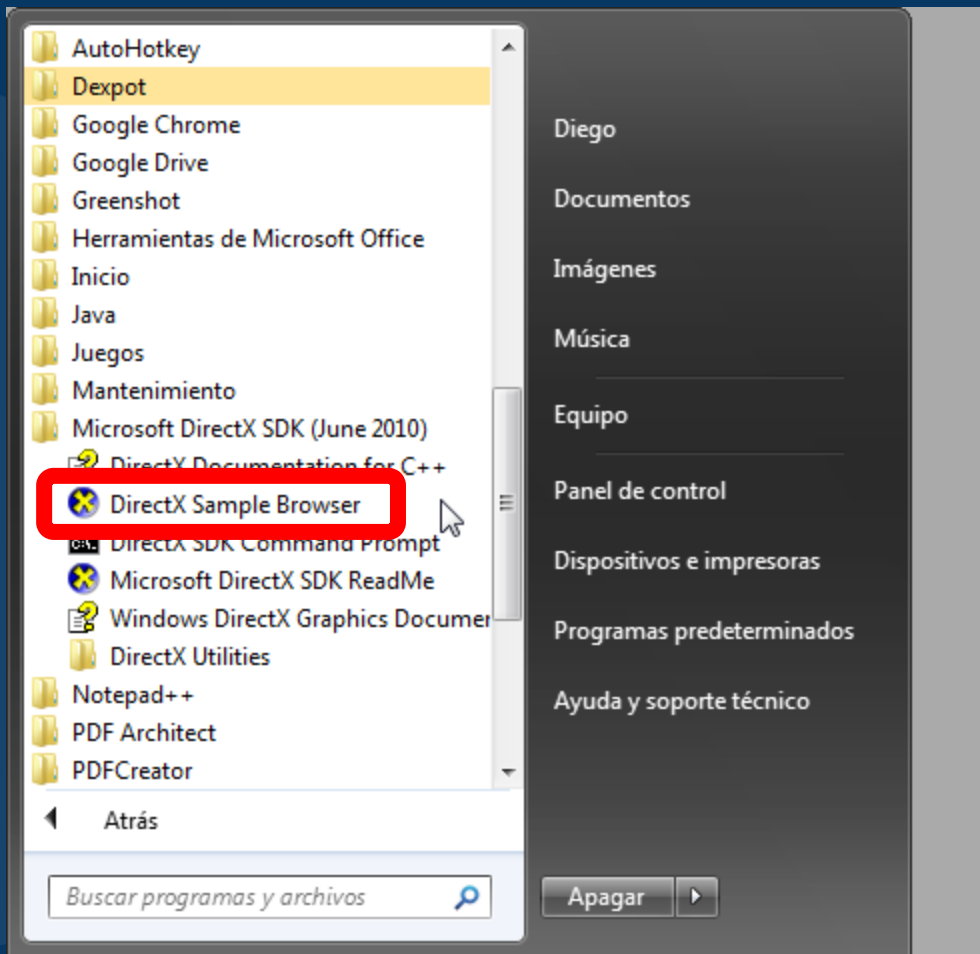


DirectX 11

First Project



Search

Clear

show

All Areas

sort by

Difficulty

 Auto-launch
documentation

Direct3D Texture Converter

utility

Textures can be converted between formats from the command line using the Texture Converter (texconv.exe). Supports Direct3D 9 texture formats.

[Executable](#) [Documentation](#)

Direct3D Extended Texture Converter

utility

This version of the Texture Converter utility supports Direct3D 10.x and 11 texture formats (texconv.exe).

[Executable](#) [Documentation](#)

Tutorial 00: Win32 Basics

C++

(December 2005)

beginner

tutorial

In this preliminary tutorial, we will go through the elements necessary to create a Win32 application. We will be setting up an empty window to prepare for Direct3D 10.

[Documentation](#) [Install Project](#)

Tutorial 01: Direct3D 10 Basics

C++

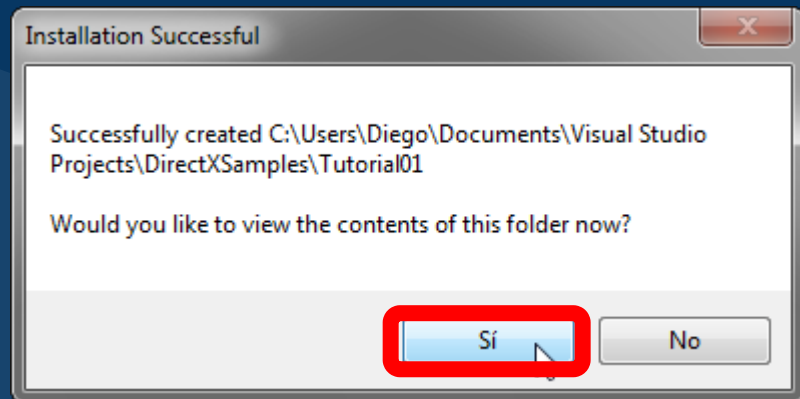
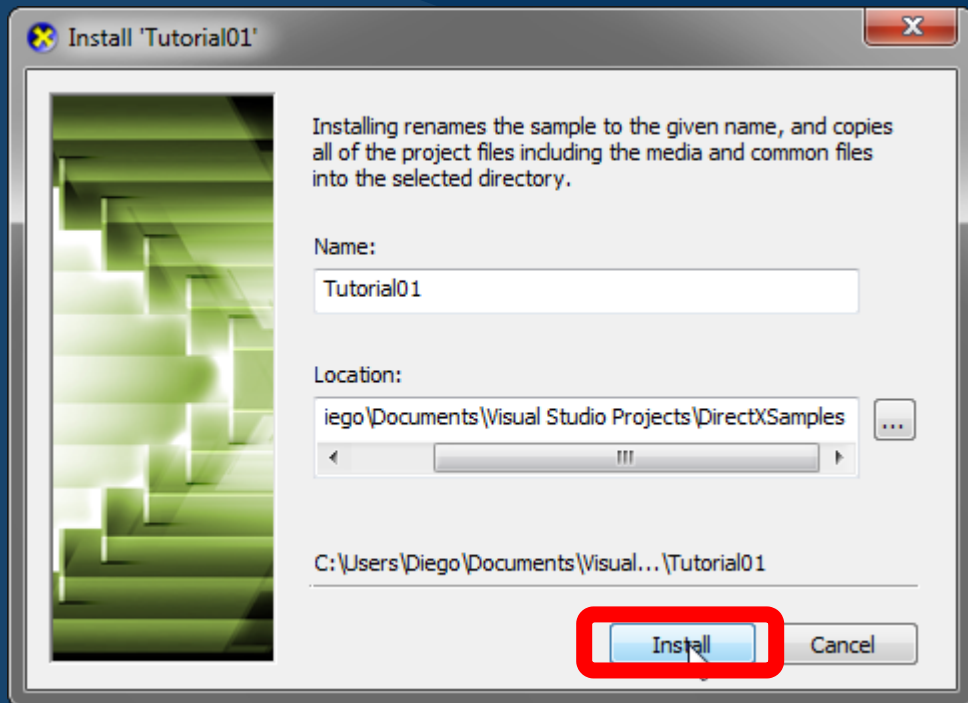
(December 2005)

beginner

tutorial

In this first tutorial, we will go through the elements necessary to create a minimal Direct3D 10 application. Every Direct3D 10 application must have these elements to function properly. The elements include setting up a window and a device object then displaying a color on the window.

[Documentation](#) [Install Project](#)



Diego > Mis documentos > Visual Studio Projects > DirectXSamples > Tutorial01

Organizar > Abrir > Compartir con > Correo electrónico > Grabar > Nueva carpeta

Buscar Tutorial01

Favoritos

- Descargas
- Escritorio
- Sitios recientes
- Bibliotecas
- Documentos
- Imágenes
- Música
- Subversion
- Videos
- Grupo en el hogar
- Equipo
- Disco local (C:)
- Unidad de DVD RW (D:) 28_1
- Mis sitios Web en MSN
- Red

Nombre	Fecha de modifica...	Tipo	Tamaño
DXUT	15/03/2015 19:53	Carpeta de archivos	
directx.ico	15/03/2015 19:53	Icono	25 KB
Resource.h	15/03/2015 19:53	C/C++ Header	1 KB
Tutorial01.cpp	15/03/2015 19:53	C++ Source	9 KB
Tutorial01.jpg	15/03/2015 19:53	Imagen JPEG	27 KB
Tutorial01.rc	15/03/2015 19:53	Archivo RC	3 KB
Tutorial01_2008.sln	15/03/2015 19:53	Microsoft Visual S...	2 KB
Tutorial01_2008.vcxproj	15/03/2015 19:53	VC++ Project	12 KB
Tutorial01_2010.sln	15/03/2015 19:53	Microsoft Visual S...	2 KB
Tutorial01_2010.vcxproj	15/03/2015 19:53	VC++ Project	18 KB
Tutorial01_2010.vcxproj.filters	15/03/2015 19:53	VC++ Project	1 KB

Abrir

- Edit with Notepad++
- Abrir con
 - Microsoft Visual Studio Express 2013 for Windows Desktop
 - Elegir programa predeterminado...
- Convert to PDF
- Convert to PDF and Email
- Compartir con
- TortoiseSVN
- WinMerge
- Añadir al archivo...
- Añadir a "Tutorial01_2010.rar"
- Añadir y enviar por email...
- Añadir a "Tutorial01_2010.rar" y enviar por email

Tutorial01_2010.sln
Microsoft Visual Studio Solution
Fecha de modifica... 15/03/2015 19:53
Tamaño: 1,58 KB

Review Project And Solution Changes



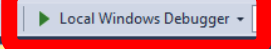
Upgrade VC++ Compiler and Libraries

The following project(s) uses an earlier version of the Visual C++ compiler and libraries. The project(s) will be upgraded to use the Microsoft Visual Studio 2013 compiler and libraries. Any managed or native code project(s) using C++/CLI extensions will be automatically upgraded to target .NET Framework 4.5. Note: If you do not upgrade the project(s), building your project(s) will require the corresponding version of Visual Studio to be installed.

..\Tutorial01\Tutorial01_2010.vcxproj

OK

Cancel



Solution Explorer

Search Solution Explorer (Ctrl+)

- Solution 'Tutorial01_2010' (1 project)
 - Tutorial01
 - DXUT
 - External Dependencies
 - Resource Files
 - Tutorial01.cpp

Properties

Tutorial01.cpp File Properties

Misc

(Name)	Tutorial01.cpp
Content	False
File Type	C/C++ Code
Full Path	C:\Users\Diego\Documents\Visual S
Included In Project	True
Relative Path	Tutorial01.cpp

(Name)
Names the file object.

```
Tutorial01 (Global Scope)
1  //-----
2  // File: Tutorial01.cpp
3  //
4  // This application demonstrates creating a Direct3D 11 device
5  //
6  // Copyright (c) Microsoft Corporation. All rights reserved.
7  //-----
8  #include <windows.h>
9  #include <d3d11.h>
10 #include <d3dx11.h>
11 #include "resource.h"
12
13
14 //-----
15 // Global Variables
16 //-----
17 HINSTANCE      g_hInst = NULL;
18 HWND          g_hWnd = NULL;
19 D3D_DRIVER_TYPE g_driverType = D3D_DRIVER_TYPE_NULL;
20 D3D_FEATURE_LEVEL g_featureLevel = D3D_FEATURE_LEVEL_11_0;
21 ID3D11Device*   g_pd3dDevice = NULL;
22 ID3D11DeviceContext* g_pImmediateContext = NULL;
23 IDXGISwapChain* g_pSwapChain = NULL;
24 ID3D11RenderTargetView* g_pRenderTargetView = NULL;
25
26
27 //-----
28 // Forward declarations
29 //-----
30 HRESULT InitWindow( HINSTANCE hInstance, int nCmdShow );
31 HRESULT InitDevice();
32 void CleanupDevice();
33 LRESULT CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );
34 void Render();
35
36
37 //-----
38 // Entry point to the program. Initializes everything and goes into a message processing
39 // loop. Idle time is used to render the scene.
40 //-----
41 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow )
```

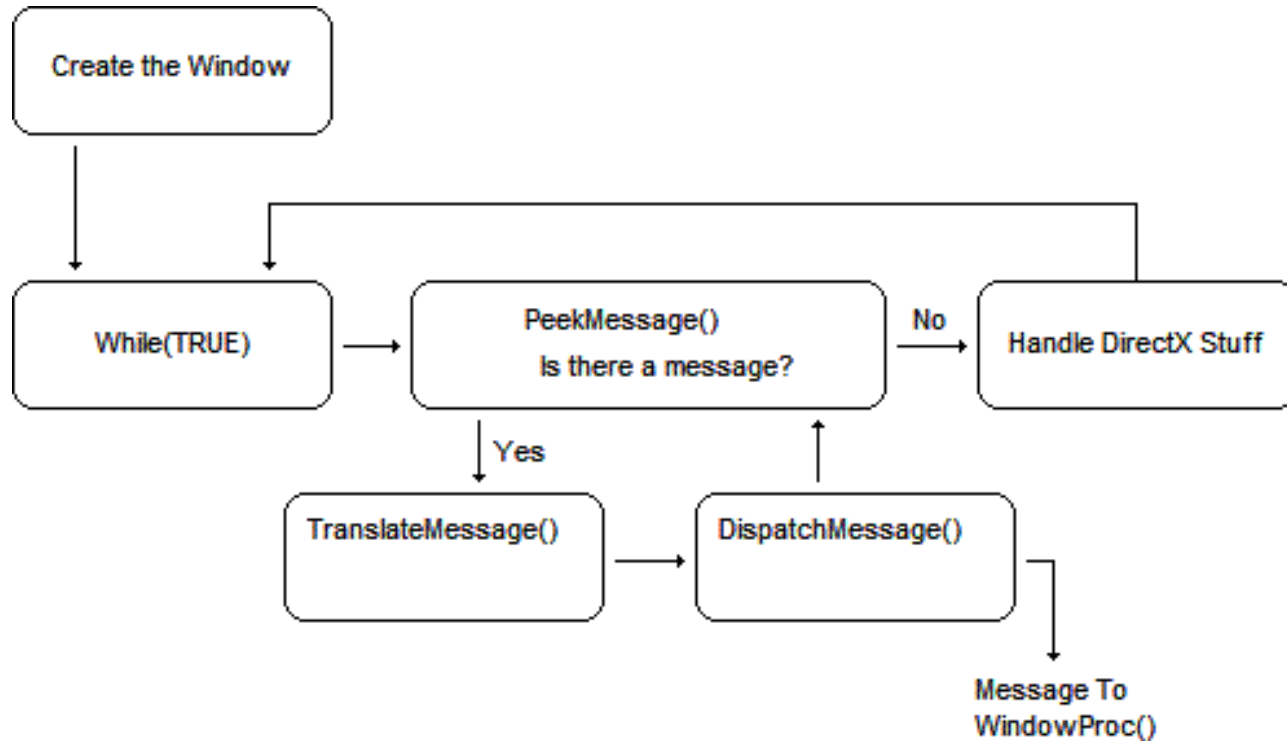
Challenge 2.1: Change cursor & window size

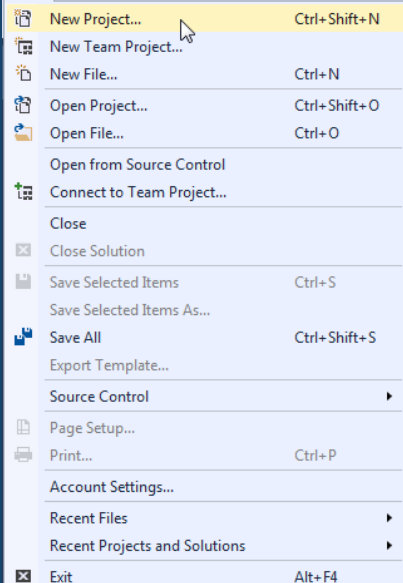
goo.gl/forms/UvamltomQM

Steps for a basic DirectX window

- Window initialization
- Application loop
- Window callback procedure
- DirectX configuration, initializing, rendering and clean up

The real-time message loop





Discover what's new in Express 2013 for Windows Desktop

You can find information about new features and enhancements in Express 2013 for Windows Desktop by reviewing the following sections.

[Learn about new features in Express 2013 for Windows Desktop](#)

[See what's new in .NET Framework 4.5.1](#)

[Explore what's new in Team Foundation Service](#)

[Connect to Azure](#)

[Learn more about Azure](#)

[Connect](#)

[Relocate the What's New information](#)

What's new in Windows Desktop

[Visual C#](#)

[Visual Basic](#)

[Visual C++](#)

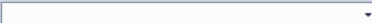
[Windows Presentation Foundation \(WPF\)](#)

[Windows Forms](#)

Solution Explorer

Output

Show output from:



Error List Output Find Symbol Results

Toolbox

New Project

Recent

Installed

- Templates
 - Visual Basic
 - Visual C#
 - Visual C++**
 - CLR
 - General
 - Test
 - Win32
 - SQL Server
 - Python
 - Visual Studio Solutions
 - Samples
 - Online

Sort by: Default

Search Installed Templates (Ctrl+E)

Icon	Project Name	Type	Description
	Win32 Console Application	Visual C++	
	Win32 Project	Visual C++	
	Empty Project	Visual C++	Type: Visual C++ An empty project for creating a local application
	Makefile Project	Visual C++	

[Click here to go online and find templates.](#)

Name: DirectX_FirstProject

Location: c:\users\diego\documents\visual studio 2013\Projects

Solution: Create new solution

Solution name: DirectX_FirstProject

Create directory for solution

Add to source control

Solution Explorer

Project (1 project)

Project Dependencies

Project Properties

DirectX_FirstProject

c:\Users\Diego\documents\visual studio 2013\Projects\DirectX_FirstProject



Output

Show output from:

(Name)
Specifies the project name.

Solution Explorer

Search Solution Explorer (Ctrl+)

- Solution 'DirectX_FirstProject' (1 project)
 - DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files

Add

- Class Wizard... Ctrl+Shift+X
- Scope to This
- New Solution Explorer View
- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V
- Delete Del
- Rename F2
- Properties

- New Item... Ctrl+Shift+A
- Existing Item... Shift+Alt+A
- New Filter
- Class... Shift+Alt+C

Properties

Source Files Filter Properties

Advanced

Parse Files	True
SCC Files	True

General

(Name)	Source Files
Filter	cpp;c;cc;co;q;def;idl;hpp;bat;asm;
Unique Identifier	{4FC737F1-C7A5-4376-A066-2A32D}

(Name)
Specifies the name of the filter.

Output

Show output from:

Error List Output Find Symbol Results

Solution Explorer

Search Solution Explorer (Ctrl+)

- Solution 'DirectX_FirstProject' (1 p)
- DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files

Properties

Source Files Filter Properties

Advanced

- Parse Files True
- SCC Files True

General

- (Name) Sour
- Filter cpp;
- Unique Identifier (4FC

(Name)
Specifies the name of the filter.

Add New Item - DirectX_FirstProject

Search Installed Templates (Ctrl+E)

Sort by: Default

Visual C++	C++ File (.cpp)	Visual C++
UI	Header File (.h)	Visual C++
Code		
Web		
Test		
Utility		
Property Sheets		
Online		

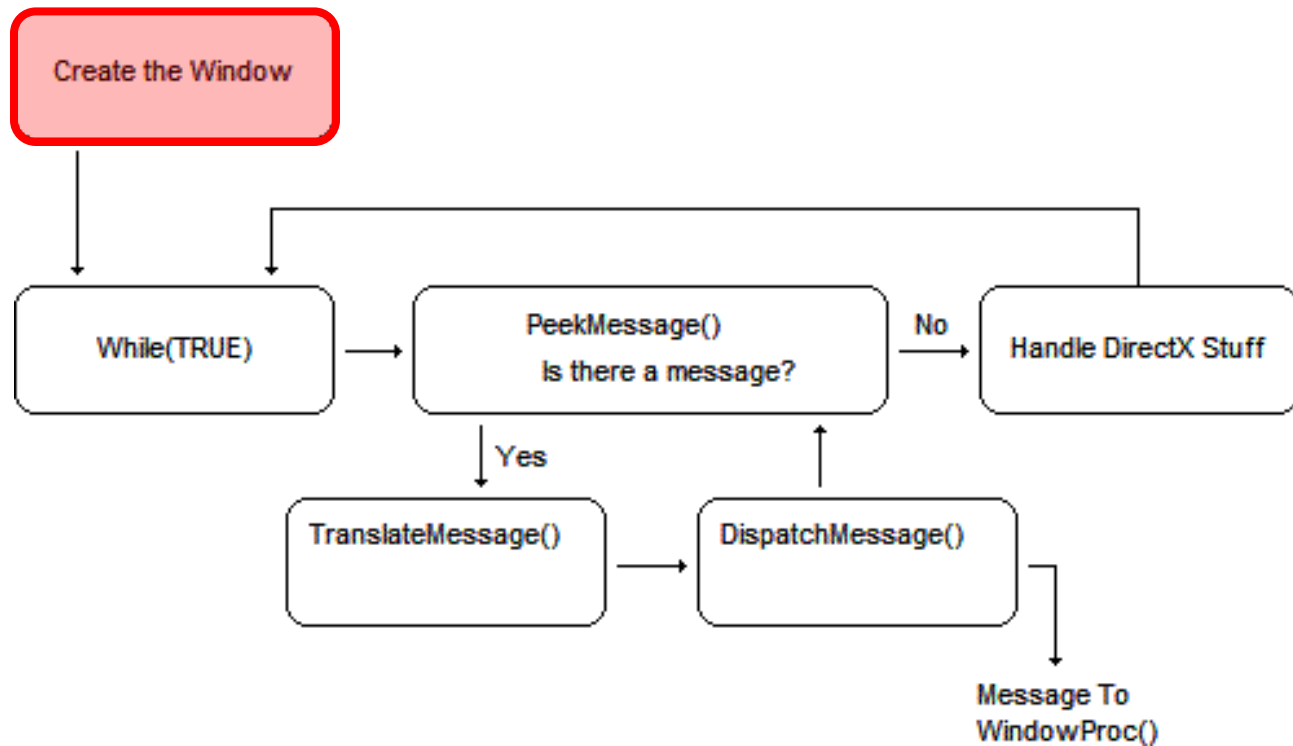
[Click here to go online and find templates.](#)

Name:

Location: Browse...

Add Cancel

Window Initialization



Solution Explorer

Search Solution Explorer (Ctrl+)

- Solution 'DirectX_FirstProject' (1 project)
 - DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - main.cpp

```

1 #include<Windows.h>
2
3 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,
4                     LPWSTR cmdLine, int cmdShow )
5 {
6     return 0;
7 }

```

Properties

wWinMain VCodeFunction

C++

(Name)	wWinMain
File	c:\users\diego\documents\visu
FullName	wWinMain
IsDefault	False
IsDelete	False
IsFinal	False
IsInjected	False

Output

Show output from:

Error List Output Find Symbol Results

WinMain parameters

hInstance. The handle of the application's current instance.

prevInstance. The handle of the previous instance of the application. This will always be NULL according to the MSDN documentation. Since this will always be NULL, if you need a way to determine whether a previous instance of the application is already running, the documentation recommends creating a uniquely named mutex using `CreateMutex`. Although the mutex will be created, the `CreateMutex` function will return `ERROR_ALREADY_EXISTS`.

WinMain parameters

cmdLine. The command line for the application without the program's name. This allows you to pass commands to the application, such as from the command prompt, by use of a shortcut with the command string provided, etc.

cmdShow. An ID that specifies how the window should be shown.

CodeShare google document

goo.gl/pb4Zwb

DirectX_FirstProject - Microsoft Visual Studio Express 2013 for Windows Desktop

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP

Local Windows Debugger | Debug | Win32

Quick Launch (Ctrl+Q) Diego de Palacio Ruiz Cabañas

Solution Explorer

Solution 'DirectX_FirstProject' (1 project)

- DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - main.cpp

Properties

wWinMain VCCodeFunction

C++

(Name)	wWinMain
File	c:\users\diego\documents\visual
FullName	wWinMain
IsDefault	False
IsDelete	False
IsFinal	False

main.cpp

```
1 #include<windows.h>
2
3 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,
4 LPWSTR cmdLine, int cmdShow )
5 {
6 #pragma region CreateWindow
7 UNREFERENCED_PARAMETER(prevInstance);
8 UNREFERENCED_PARAMETER(cmdLine);
9
10 WNDCLASSEX wndClass = { 0 };
11 wndClass.cbSize = sizeof(WNDCLASSEX);
12 wndClass.style = CS_HREDRAW | CS_VREDRAW;
13 wndClass.lpfnWndProc = WndProc;
14 wndClass.hInstance = hInstance;
15 wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
16 wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
17 wndClass.lpszMenuName = NULL;
18 wndClass.lpszClassName = "DirectX11_FirstWindow";
19
20 if (!RegisterClassEx(&wndClass))
21 return -1;
22
23 RECT rc = { 0, 0, 640, 480 };
24 AdjustWindowRect(&rc, WS_OVERLAPPEDWINDOW, FALSE);
25
26 HWND hwnd = CreateWindowA("DirectX11_FirstWindow", "First Window",
27 WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, rc.right - rc.left,
28 rc.bottom - rc.top, NULL, NULL, hInstance, NULL);
29
30 if (!hwnd)
31 return -1;
32
33 ShowWindow(hwnd, cmdShow);
34 #pragma endregion CreateWindow
35
36 return 0;
37 }
```

Copy code block
contained on page 1

Window Initialization

`UNREFERENCED_PARAMETER` can be used to avoid compiler warnings about parameters that are unused by a function's body.

Window Class

```
typedef struct tagWNDCLASSEX {  
    UINT cbSize;  
    UINT style;  
  
    WNDPROC lpfnWndProc;  
    int cbClsExtra;  
    int cbWndExtra;  
    HINSTANCE hInstance;  
    HICON hIcon;  
    HCURSOR hCursor;  
    HBRUSH hbrBackground;  
    LPCTSTR lpszMenuName;  
    LPCTSTR lpszClassName;  
    HICON hIconSm;  
} WNDCLASSEX, *PWNDCLASSEX;
```

cbSize. The size of the structure in bytes.

style. Style flags used to define the window's look.

lpfnWndProc. A callback function that is called whenever an event notification comes from the operating system.

cbClsExtra. Number of extra bytes to allocate for the window structure.

cbWndExtra. Number of extra bytes to allocate for the window's instance.

Window Class

hInstance. The application instance that contains the windows procedure (callback) for this window class.

hIcon. The resource ID for the icon graphic to be displayed for the application.

hCursor. The resource ID for the graphic that will act as the cursor.

hbrBackground. A handle to the background brush that will be used for painting the window's background.

lpszMenuName. null-terminated string of the resource name for the menu.

lpszClassName. null-terminated string for what you wish to name your window class.

hIconSm. Handle to the window's small icon

Window Initialization

RegisterClassEx is used to register the window class.

AdjustWindowRect to calculate the size required of the window based on our desired dimensions and style. The type of window will determine how much true size we'll need, taking on account the client and non-client area.

AdjustWindowRect function takes a rectangle that defines the dimensions of the window, the window style flag of the window being created and a Boolean indicating whether or not the window has a menu, which affects the non-client area.

Window Initialization

CreateWindow(A) takes as parameters:

lpClassName (optional)—The window class name (same name used for the window class structure).

lpWindowName (optional)—The window title bar text.

dwStyle—The window's style flags.

X—The window's horizontal position.

Y—The window's vertical position.

nWidth—The window's width.

hHeight—The window's height.

hWndParent (optional)—Handle to the parent window's handle (optional if this new window is a pop-up or child window).

hMenu (optional)—Resource handle to the window's menu.

Window Initialization

hInstance (optional)—The application instance ID (first parameter of `wWinMain`).

lpParam (optional)—Data to be passed to the window and made available via the **lpParam** parameter of the windows proc callback function.

ShowWindow, takes as parameters the window handle returned by **CreateWindow(A)** and the command show flag.

show

Direct3D 11

sort by

Date

 Auto-launch
documentation

Tutorial 01: Direct3D 11 Basics C++ (June 2010)

new

beginner

tutorial

In this first tutorial, we will go through the elements necessary to create a minimal Direct3D 11 application. Every Direct3D 11 application must have these elements to function properly. The elements include setting up a window and a device object then displaying a color on the window.

[Documentation](#) [Install Project](#)

Tutorial 02: Rendering a Triangle C++ (June 2010)

new

beginner

tutorial

In the previous tutorial, we built a minimal bare bone Direct3D 11 application that outputs a single color to the window. In this tutorial, we will extend the application to render a single triangle on the screen. We will go through the process to setup the data structures associated with a triangle.

[Documentation](#) [Install Project](#)

Tutorial 03: Shaders C++ (June 2010)

new

beginner

tutorial

In the previous tutorial, we setup a vertex buffer and passed one triangle to the GPU. Now, we will actually step through the graphics pipeline and look at how each stage works. The concept of shaders and the effect system will be explained.

[Documentation](#) [Install Project](#)

Tutorial 04: 3D Spaces C++ (June 2010)

new

beginner

tutorial

In the previous tutorial, we have successfully rendered a triangle in the center of our application window. We haven't paid much attention to the vertex positions that we have picked in our vertex buffer. In this tutorial, we will delve into the details of 3D positions and transformation.

Direct3D 11 Tutorial 1: Direct3D 11 Basics

Microsoft Visual Studio Express 2013 for Windows De...

This project is out of date:

Tutorial01 - Debug Win32

Would you like to build it?

Yes

No

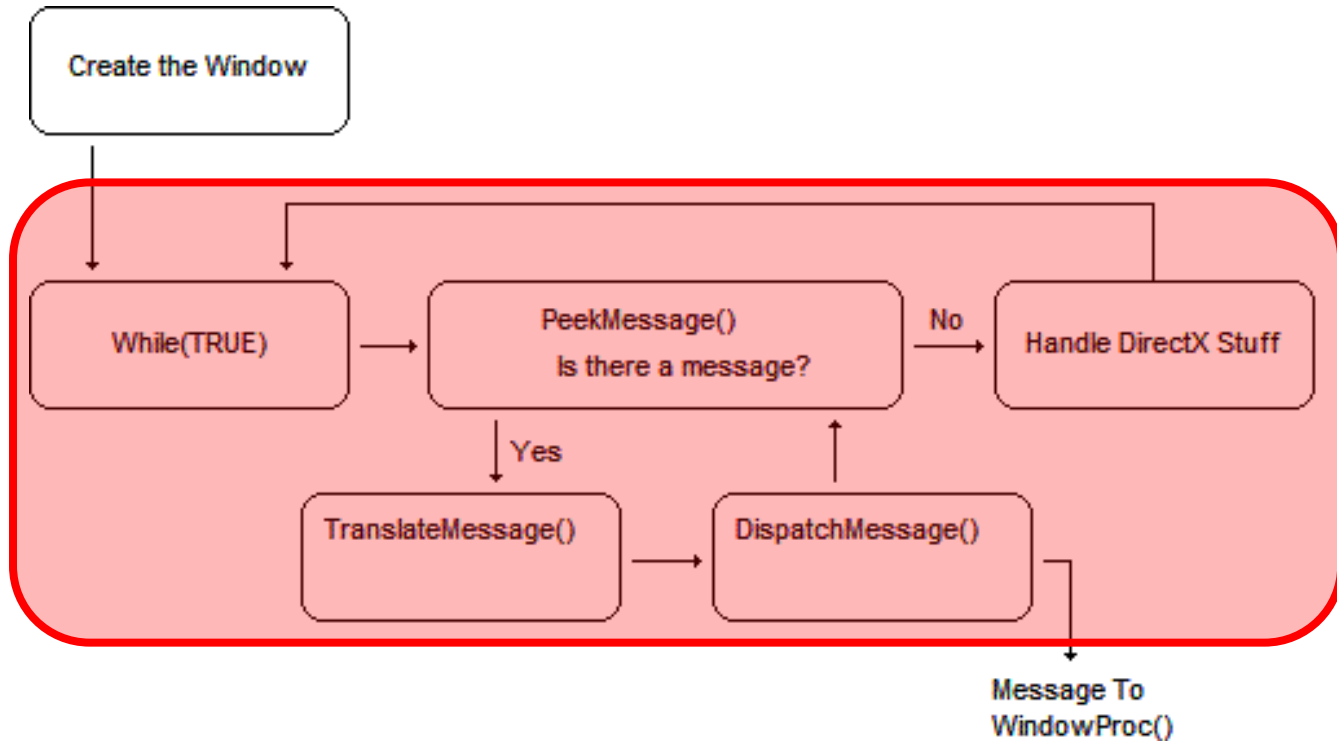
Cancel

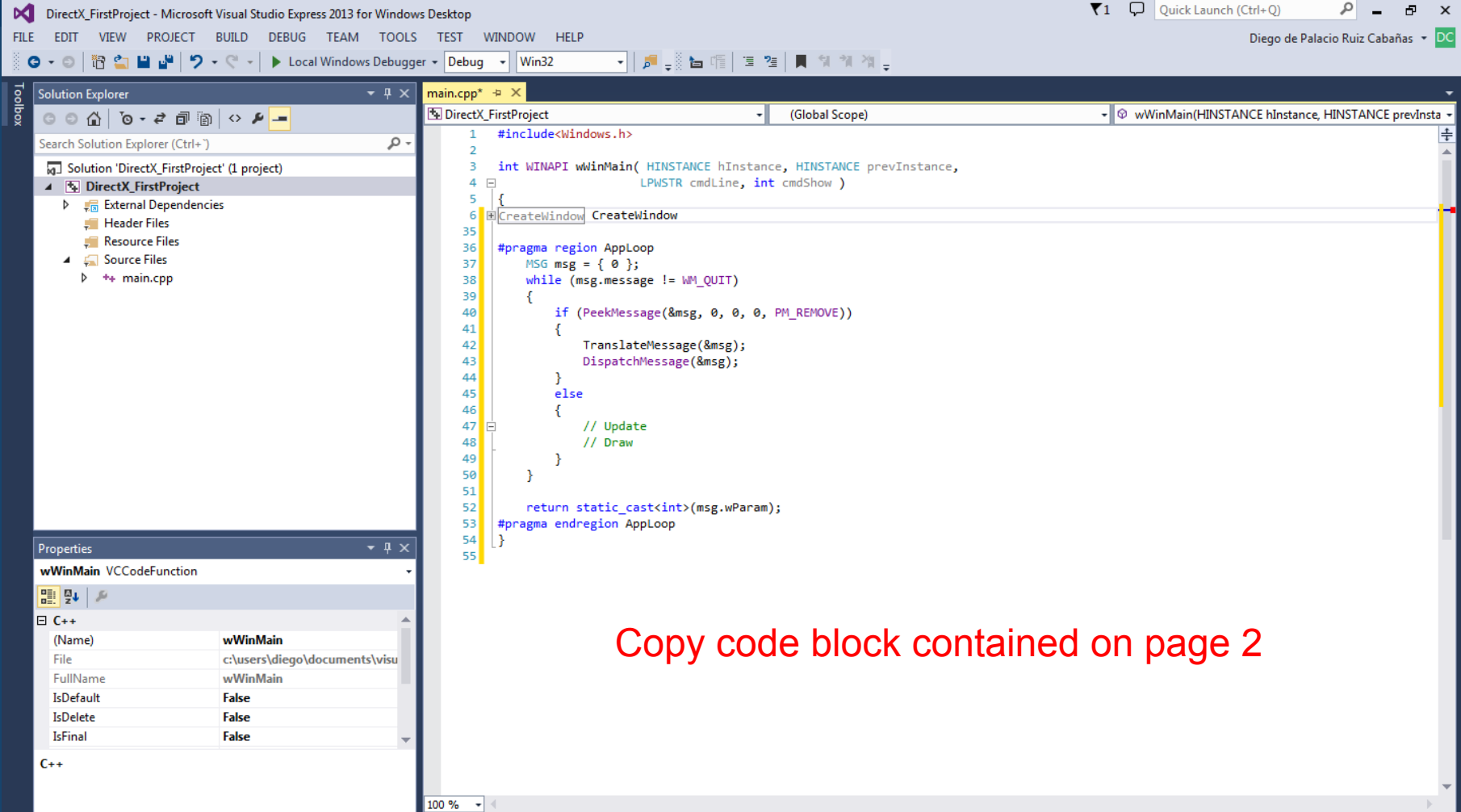
Do not show this dialog again

Challenge 2.2: Change background color

goo.gl/forms/6oZoWKxqKv

Application Loop





Copy code block contained on page 2

Application Loop

Win32 GUI applications are ***event-based*** applications. This essentially means that when an event happens, the application is notified of it, and some action then occurs.

In video games, the applications are ***real-time***, meaning that whether or not some event or action takes place will not keep the application from performing many tasks throughout its lifetime.

Both real-time and event-based programs must run until the user decides to quit. This introduces the concept of the application loop.

Application Loop

MSG is a Win32 structure used to hold window messages, some of which come from the operating system, and it is up to the application to respond to these messages. If this does not happen after a certain amount of time has passed, the operating system will report the application as not responding.

Application Loop

With window messages we need to process them, and dispatch (respond) to these messages.

PeekMessage Win32 function retrieves a message for the associated window. The first parameter is the structure that will hold the message (its address), the window handle (optional), min and max message filter flags (optional), and the remove flag. Specifying `PM_REMOVE` as the remove flag like we've done removes it from the queue.

If there is a message obtained by **PeekMessage**, we can respond to that message by calling **TranslateMessage** and **DispatchMessage**.

Application Loop

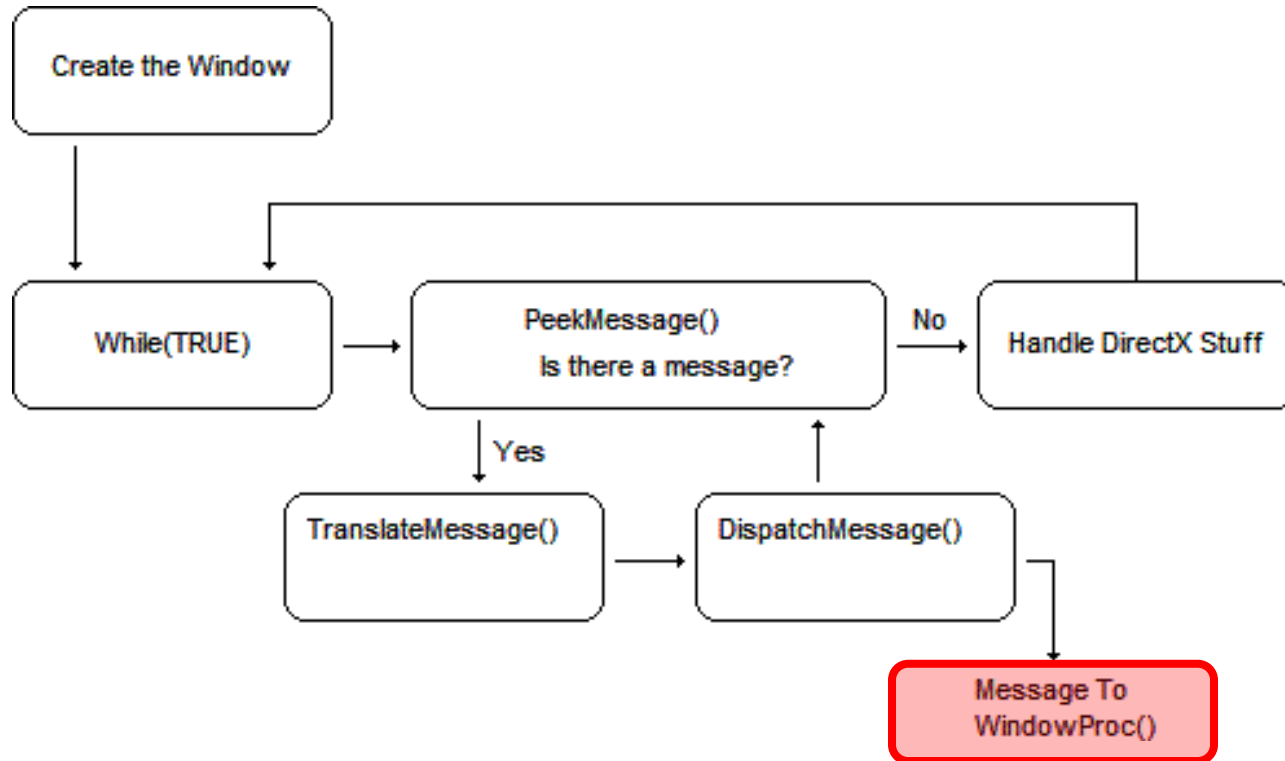
If there are no messages, the only thing to do is to perform game updates and rendering.

In this part of the code (now only commented), a series of game-specific steps are taken for each rendered frame.

Most games strive to reach 30 or 60 frames per second, or in other words 30 to 60 game loop iterations for each second of real-world time.

The last line of code in the **wWinMain** function returns 0. Generally this function is returning an exit code, which would only matter if was launched the application from another application.

Windows Callback Procedure



Solution Explorer

Search Solution Explorer (Ctrl+)

Solution 'DirectX_FirstProject' (1 project)

DirectX_FirstProject

External Dependencies

Header Files

Resource Files

Source Files

main.cpp

Properties



main.cpp* x

DirectX_FirstProject

(Global Scope)

WndProc(HWND hwnd, UINT message, WPARAM wParam

```
1 #include<windows.h>
2
3 LRESULT CALLBACK WndProc( HWND hwnd, UINT message,
4     WPARAM wParam, LPARAM lParam );
5
6 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,
7     LPWSTR cmdLine, int cmdShow )
8 {
9     #pragma region CreateWindow
10     UNREFERENCED_PARAMETER(prevInstance);
11     UNREFERENCED_PARAMETER(cmdLine);
12
13     WNDCLASSEX wndClass = { 0 };
14     wndClass.cbSize = sizeof(WNDCLASSEX);
15     wndClass.style = CS_HREDRAW | CS_VREDRAW;
16     wndClass.lpfnWndProc = WndProc;
17     wndClass.hInstance = hInstance;
18     wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
19     wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
20     wndClass.lpszMenuName = NULL;
21     wndClass.lpszClassName = "DirectX11_FirstWindow";
22
23     if (!RegisterClassEx(&wndClass))
24         return -1;
25
26     RECT rc = { 0, 0, 640, 480 };
```

150 %

Windows Callback Procedure

The **WndProc** function is a callback function, meaning that it is called whenever messages are being obtained and processed by our application.

DirectX_FirstProject - Microsoft Visual Studio Express 2013 for Windows Desktop

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP

Local Windows Debugger Debug Win32

Quick Launch (Ctrl+Q) Diego de Palacio Ruiz Cabañas

Solution Explorer

Search Solution Explorer (Ctrl+)

Solution 'DirectX_FirstProject' (1 project)

- DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - main.cpp

Properties

WndProc VCCodeFunction

C++

(Name)	WndProc
File	c:\users\diego\documents\visu
FullName	WndProc
IsDefault	False
IsDelete	False
IsFinal	False

C++

main.cpp

```
1 #include<Windows.h>
2
3 LRESULT CALLBACK WndProc( HWND hwnd, UINT message,
4                          WPARAM wParam, LPARAM lParam );
5
6 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,
7                    LPWSTR cmdLine, int cmdShow )
8 {
9     CreateWindow CreateWindow
38
39     AppLoop AppLoop
57 }
58
59 LRESULT CALLBACK WndProc( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam )
60 {
61     PAINTSTRUCT paintStruct;
62     HDC hDC;
63     switch (message)
64     {
65     case WM_PAINT:
66         hDC = BeginPaint(hwnd, &paintStruct);
67         EndPaint(hwnd, &paintStruct);
68         break;
69     case WM_DESTROY:
70         PostQuitMessage(0);
71         break;
72     default:
73         return DefWindowProc(hwnd, message, wParam, lParam);
74     }
75     return 0;
76 }
```

Copy code block contained on page 3

Windows Callback Procedure

The callback function takes as parameters the handle of the window dispatching this message, the message as an unsigned integer, and two parameters specifying additional information (wParam and lParam). The last two parameters are used to supply data to the callback function for messages that require more data to perform some type of action.

The two messages that most Win32 applications handle are WM_PAINT (sent when Windows would like the window to be redrawn) and WM_DESTROY (sent when the window is being destroyed).

Windows Callback Procedure

An important thing to note is that any message you don't process in the switch statement goes into **DefWindowProc**, which defines the default behavior for every Windows message. Anything not processed needs to go into **DefWindowProc** for the application to behave correctly.

The paint message is handled by calling Win32 functions to draw the window's background, which is handled by calling **BeginPaint** and **EndPaint**. Since Direct3D will be doing all of our rendering, this will be all we'll have to do for this message

Windows Callback Procedure

The quit message is handled by calling the Win32 `PostQuitMessage` function, which will cause the MSG object in our application loop to retrieve a `WM_QUIT` message, which causes the application loop to end and the application to quit as we've intended. Since there are situations where we don't want to just flat-out quit, we can use this to only post a quit message if we really want to quit.

Solution Explorer

Search Solution Explorer (Ctrl+)

- Solution 'DirectX_FirstProject' (1 project)
 - DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - main.cpp

Properties

wWinMain VCCodeFunction

C++

(Name)	wWinMain
File	c:\users\diego\documents\visu
FullName	wWinMain
IsDefault	False
IsDelete	False
IsFinal	False

C++

```

main.cpp* - X
DirectX_FirstProject (Global Scope) wWinMain(HINSTANCE hInstance, HINSTANCE prevInsta
1 #include<Windows.h>
2
3 LRESULT CALLBACK WndProc( HWND hwnd, UINT message,
4     WPARAM wParam, LPARAM lParam );
5
6 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,
7     LPWSTR cmdLine, int cmdShow )
8 {
9 #pragma region CreateWindow
10 UNREFERENCED_PARAMETER(prevInstance);
11 UNREFERENCED_PARAMETER(cmdLine);
12
13 WNDCLASSEX wndClass = { 0 };
14 wndClass.cbSize = sizeof(WNDCLASSEX);
15 wndClass.style = CS_HREDRAW | CS_VREDRAW;
16 wndClass.lpfnWndProc = WndProc;
17 wndClass.hInstance = hInstance;
18 wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
19 wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
20 wndClass.lpszMenuName = NULL;
21 wndClass.lpszClassName = "DirectX11_FirstWindow";
22
23 if (!RegisterClassEx(&wndClass))
24     return -1;
25
26 RECT rc = { 0, 0, 640, 480 };
27 AdjustWindowRect(&rc, WS_OVERLAPPEDWINDOW, FALSE);
28
29 HWND hwnd = CreateWindowA("DirectX11_FirstWindow", "First Window",
30     WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, rc.right - rc.left,
31     rc.bottom - rc.top, NULL, NULL, hInstance, NULL);
32
33 if (!hwnd)
34     return -1;
35
36 ShowWindow(hwnd, cmdShow);
37 #pragma endregion CreateWindow
38
39 #pragma region AppLoop
40 MSG msg = { 0 };
41 while (msg.message != WM_QUIT)
    
```

Microsoft Visual Studio Express 2013 for Windows De...



This project is out of date:

DirectX_FirstProject - Debug Win32

Would you like to build it?

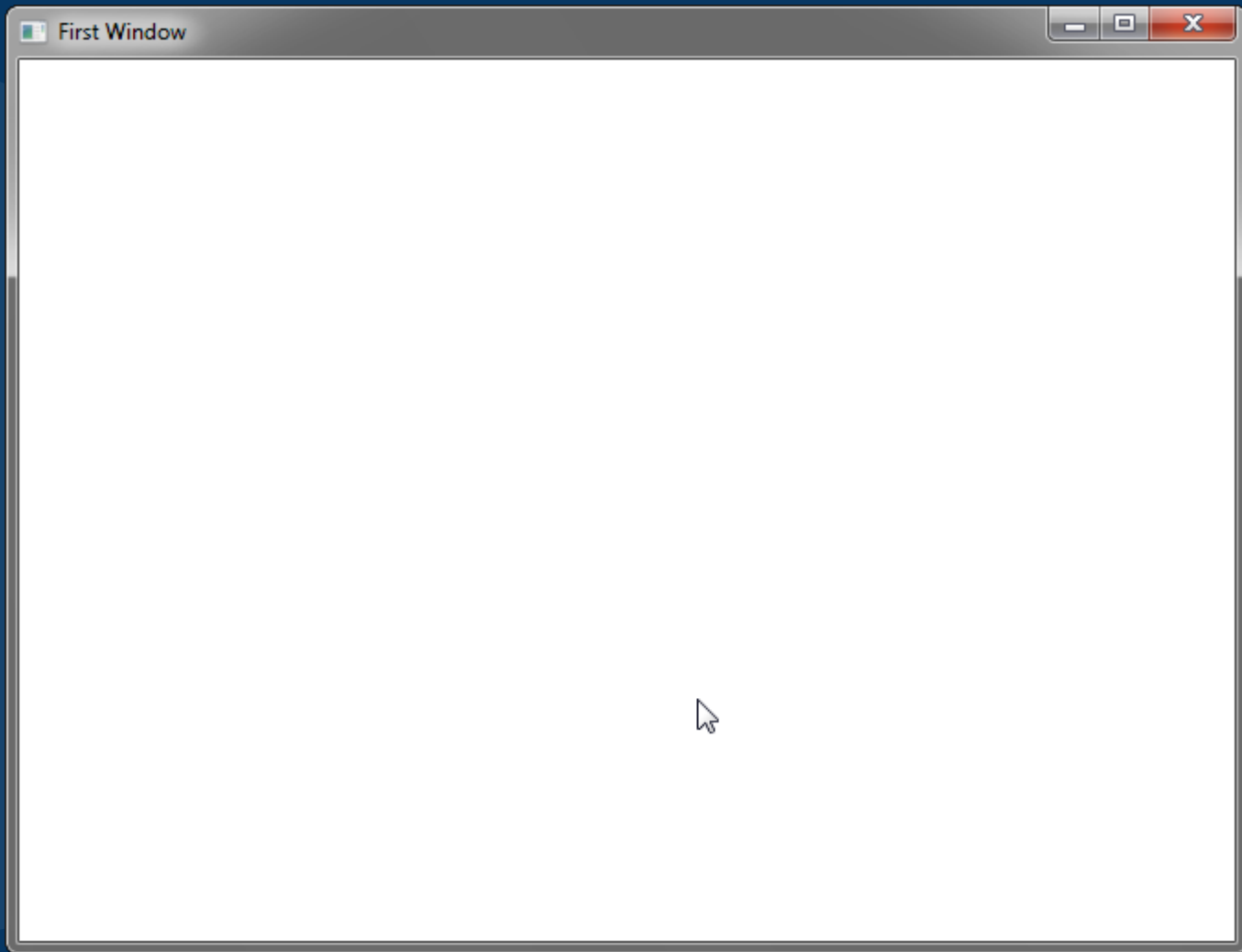
Yes

No

Cancel



Do not show this dialog again



Challenge 2.3: First DirectX window

goo.gl/forms/xnfGcr1yAs

DirectX configuration

Solution Explorer

Search Solution Ex

Solution 'Dire

DirectX

- Extern
- Head
- Resou
- Sourc
- DirectX_FirstProject Properties... Alt+F7

Properties

wWinMain VCCodeFunction

C++

(Name)	wWinMain
File	c:\users\diego\documents\visu
FullName	wWinMain
IsDefault	False
IsDelete	False
IsFinal	False

C++

```

DirectX_FirstProject (Global Scope) wWinMain(HINSTANCE hInstance, HINSTANCE prevInsta
#include<windows.h>

LRESULT CALLBACK WndProc( HWND hwnd, UINT message,
                        WPARAM wParam, LPARAM lParam );

int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,
                    LPWSTR cmdLine, int cmdShow )
{
#pragma region CreateWindow
    UNREFERENCED_PARAMETER(prevInstance);
    UNREFERENCED_PARAMETER(cmdLine);

    WNDCLASSEX wndClass = { 0 };
    wndClass.cbSize = sizeof(WNDCLASSEX);
    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hInstance = hInstance;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wndClass.lpszMenuName = NULL;
    wndClass.lpszClassName = "DirectX11_FirstWindow";

    if (!RegisterClassEx(&wndClass))
        return -1;

    RECT rc = { 0, 0, 640, 480 };
    AdjustWindowRect(&rc, WS_OVERLAPPEDWINDOW, FALSE);

    HWND hwnd = CreateWindowA("DirectX11_FirstWindow", "First Window",
                              WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, rc.right - rc.left,
                              rc.bottom - rc.top, NULL, NULL, hInstance, NULL);

    if (!hwnd)
        return -1;

    ShowWindow(hwnd, cmdShow);
#pragma endregion CreateWindow

#pragma region AppLoop
    MSG msg = { 0 };
    while (msg.message != WM_QUIT)
    
```


Solution Explorer

Search Solution Explorer (Ctrl+)

Solution 'DirectX_FirstProject' (1 project)

- DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - main.cpp

```

1 #include<Windows.h>
2
3 LRESULT CALLBACK WndProc( HWND hwnd, UINT message,
4     WPARAM wParam, LPARAM lParam );
5
6 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,

```

DirectX_FirstProject Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Active(Debug)
 - Dependencies
 - Default Libraries
 - Specific Default Libraries
 - Module Definition File
 - Add Module to Assembly
 - Embed Managed Resource File
 - Force Symbol References
 - Delay Loaded DLLs
 - Assembly Link Resource
- Additional Dependencies

Specifies additional items to add to the link command line [i.e. kernel32.lib]

Buttons: Aceptar Cancelar Aplicar

Properties

wWinMain VCCodeFunction

C++	
(Name)	wWinMain
File	c:\users\diego\documents\visu
FullName	wWinMain
IsDefault	False
IsDelete	False
IsFinal	False

```

35
36 ShowWindow(hwnd, cmdShow);
37 #pragma endregion CreateWindow
38
39 #pragma region AppLoop
40 MSG msg = { 0 };
41 while (msg.message != WM_QUIT)

```

Solution Explorer

Search Solution Explorer (Ctrl+)

- Solution 'DirectX_FirstProject' (1 project)
 - DirectX_FirstProject
 - External Dependencies
 - Header Files
 - Resource Files
 - Source Files
 - main.cpp

```
main.cpp
DirectX_FirstProject (Global Scope)
wWinMain(HINSTANCE hInstance, HINSTANCE prevInsta
1 #include<Windows.h>
2
3 LRESULT CALLBACK WndProc( HWND hwnd, UINT message,
4                             WPARAM wParam, LPARAM lParam );
5
6 int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE prevInstance,
```

DirectX_FirstProject Property Pages

Configuration: All Configurations Platform: Active(Win32) Configuration Manager...

- General
- Debugging
- VC++ Directories
- C/C++
- Linker
 - General
 - Input
 - Manifest File
 - Debugging
 - System
 - Optimization
 - Embedded IDL
 - Windows Metadata
 - Advanced
 - All Options
 - Command Line
 - Manifest Tool

Additional Dependencies: b:\odbc32.lib;odbc32.lib;%AdditionalDependencies%

Ignore All Default Libraries <Edit...>

Ignore Specific Default Libraries

Module Definition File

Add Module to Assembly

Embed Managed Resource File

Force Symbol References

Delay Loaded DLLs

Assembly Link Resource

Additional Dependencies
Specifies additional items to add to the link command line [i.e. kernel32.lib]

Acceptar Cancelar Aplicar

Properties

wWinMain VCCodeFunction

C++

(Name)	wWinMain
File	c:\users\diego\documents\visu
FullName	wWinMain
IsDefault	False
IsDelete	False
IsFinal	False

C++

```
35
36 ShowWindow(hwnd, cmdShow);
37 #pragma endregion CreateWindow
38
39 #pragma region AppLoop
40 MSG msg = { 0 };
41 while (msg.message != WM_QUIT)
```

Configuration:

All Configurations

Platform:

Active(Win32)

Configuration Manager...

Linker

General

Input

Manifest File

Debugging

System

Optimization

Embedded IDL

Windows Metadata

Advanced

All Options

Command Line

Manifest Tool

XML Document Genera

Browse Information

Build Events

Custom Build Step

Additional Dependencies

d3d11.lib;d3dx11.lib;dxerr.lib;kernel32.lib;user32.lib;gdi32

Ignore All Default Libraries

Ignore Specific Default Libraries

Module Definition File

Add Module to Assembly

Embed Managed Resource File

Force Symbol References

Delay Loaded DLLs

Assembly Link Resource

Additional Dependencies

Specifies additional items to add to the link command line [i.e. kernel32.lib]

Aceptar

Cancelar

Aplicar

Configuration:

All Configurations

Platform:

Active(Win32)

Configuration Manager...

- ▶ Common Properties
- ▲ Configuration Properties
 - General
 - Debugging
 - VC++ Directories
 - ▶ C/C++
 - ▶ Linker
 - ▶ Manifest Tool
 - ▶ XML Document Generator
 - ▶ Browse Information
 - ▶ Build Events
 - ▶ Custom Build Step
 - ▶ Code Analysis

▲ General

Executable Directories	\$(VC_ExecutablePath_x86);\$(WindowsSDK_ExecutablePath)
Include Directories	\$(DXSDK_DIR)Include;\$(VC_IncludePath);\$(WindowsSD
Reference Directories	\$(VC_ReferencesPath_x86);
Library Directories	\$(VC_LibraryPath_x86);\$(WindowsSDK_LibraryPath_x86);
Library WinRT Directories	\$(WindowsSDK_MetadataPath);
Source Directories	\$(VC_SourcePath);
Exclude Directories	\$(VC_IncludePath);\$(WindowsSDK_IncludePath);\$(MSBuild

Include Directories

Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

Aceptar

Cancelar

Aplicar

Configuration: All Configurations

Platform: Active(Win32)

Configuration Manager...

- ▶ Common Properties
- ▲ Configuration Properties
 - General
 - Debugging
 - VC++ Directories
 - ▶ C/C++
 - ▶ Linker
 - ▶ Manifest Tool
 - ▶ XML Document Generator
 - ▶ Browse Information
 - ▶ Build Events
 - ▶ Custom Build Step
 - ▶ Code Analysis

▲ General

Executable Directories	\$(VC_ExecutablePath_x86);\$(WindowsSDK_ExecutablePath)
Include Directories	\$(DXSDK_DIR)Include;\$(VC_IncludePath);\$(WindowsSD
Reference Directories	\$(VC_ReferencesPath_x86);
Library Directories	\$(DXSDK_DIR)Lib\x86;\$(VC_LibraryPath_x86);\$(Window
Library WinRT Directories	\$(WindowsSDK_MetadataPath);
Source Directories	\$(VC_SourcePath);
Exclude Directories	\$(VC_IncludePath);\$(WindowsSDK_IncludePath);\$(MSBuild

Library Directories

Path to use when searching for library files while building a VC++ project. Corresponds to environment variable LIB.

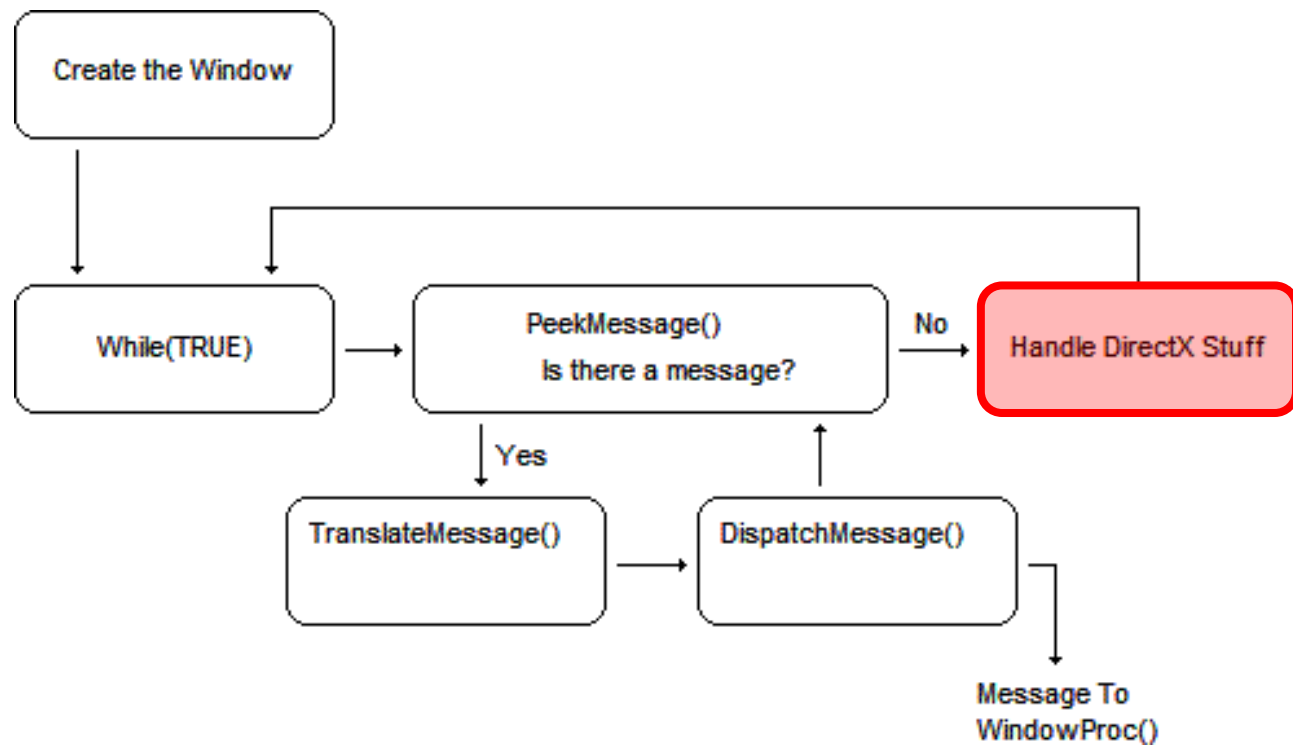
Aceptar

Cancelar

Aplicar

Execute the app now!

Initializing DirectX



Initializing DirectX

To set up Direct3D we need to complete the following steps:

1. Define the device types and feature levels we want to check for.
2. Create the Direct3D device, rendering context, and swap chain.
3. Create the render target.
4. Set the viewport.

Initializing DirectX

1. Define the device types and feature levels

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

In Direct3D 11 we can have a

- Hardware device,
- Software driver device,
- Reference device,
- WARP device.

Initializing DirectX

1. Define the device types and feature levels

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

Hardware device is a Direct3D device that runs on the graphics hardware and is the fastest of all devices.

Reference device is installed with 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:

- To test code your hardware does not support.
- 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.

Initializing DirectX

1. Define the device types and feature levels

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

Software driver device allows developers to write their own software rendering driver and use it with Direct3D. This is called a pluggable software driver.

WARP device creates a high performance Direct3D 10.1 software driver. WARP stands for Windows Advanced Rasterization Platform. We are not interested in this because it does not support Direct3D 11. Uses the Windows Graphic runtime found in Windows Vista and Windows 7 and highly optimized instructions and code.

Initializing DirectX

1. Define the device types and feature levels

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

```
D3D_DRIVER_TYPE driverTypes[] =
{
    D3D_DRIVER_TYPE_HARDWARE, D3D_DRIVER_TYPE_WARP,
    D3D_DRIVER_TYPE_REFERENCE, D3D_DRIVER_TYPE_SOFTWARE
};
```

```
unsigned int totalDriverTypes = ARRAYSIZE( driverTypes );
```

```
D3D_FEATURE_LEVEL featureLevels[] =
{
    D3D_FEATURE_LEVEL_11_0,
    D3D_FEATURE_LEVEL_10_1,
    D3D_FEATURE_LEVEL_10_0
};
```

Challenge 2.4: Exit app message

goo.gl/forms/HmZFLn1U8U

Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

Each device has at least one swap chain, but multiple swap chains can be created for multiple devices. A rendering destination can be a color buffer that is rendered to and displayed to the screen, a depth buffer, a stencil buffer, and so forth.

Usually in games we have two color buffers that are being rendered onto called the primary buffer and the secondary buffer, as well as being known as the front and back buffers. The primary buffer (front buffer) is the one that is displayed to the screen, while the secondary buffer (back buffer) is being drawn to for the next frame.

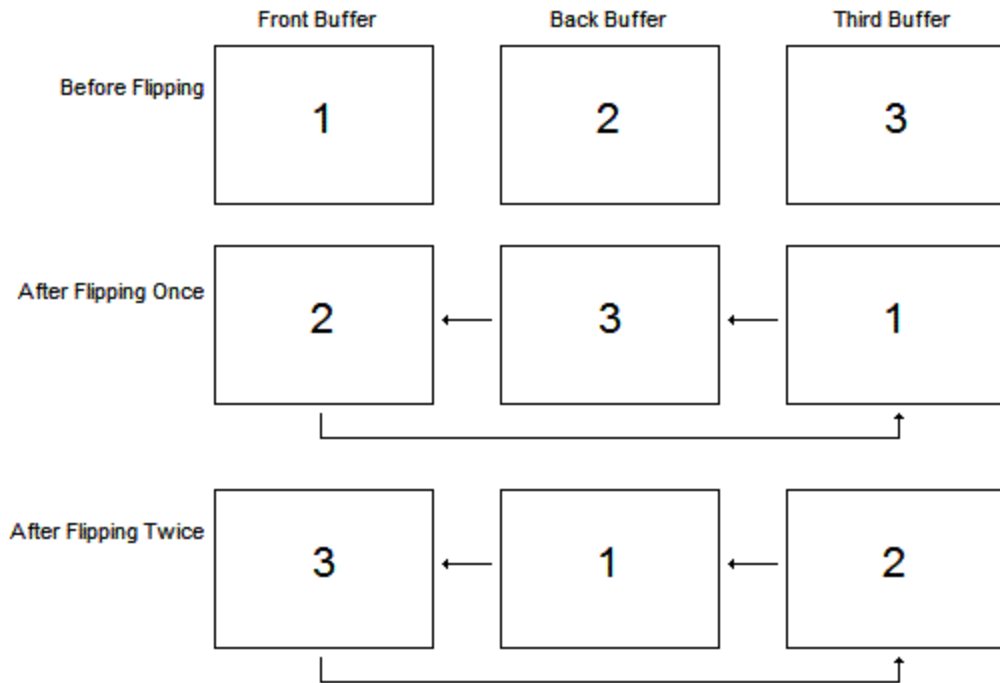
Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.



Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

```
DXGI_SWAP_CHAIN_DESC swapChainDesc;  
ZeroMemory( &swapChainDesc, sizeof( swapChainDesc ) );  
swapChainDesc.BufferCount = 1;  
swapChainDesc.BufferDesc.Width = width;  
swapChainDesc.BufferDesc.Height = height;  
swapChainDesc.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;  
swapChainDesc.BufferDesc.RefreshRate.Numerator = 60;  
swapChainDesc.BufferDesc.RefreshRate.Denominator = 1;  
swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;  
swapChainDesc.OutputWindow = hwnd;  
swapChainDesc.Windowed = true;  
swapChainDesc.SampleDesc.Count = 1;  
swapChainDesc.SampleDesc.Quality = 0;
```


Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

The sample description defines the multisampling properties of Direct3D.

Multisampling is a technique used to sample and average rendered pixels to create smoother transitions between sharp color changes. The artifacts we are attempting to reduce with multisampling are called jagged edges, also known as the staircase effect.

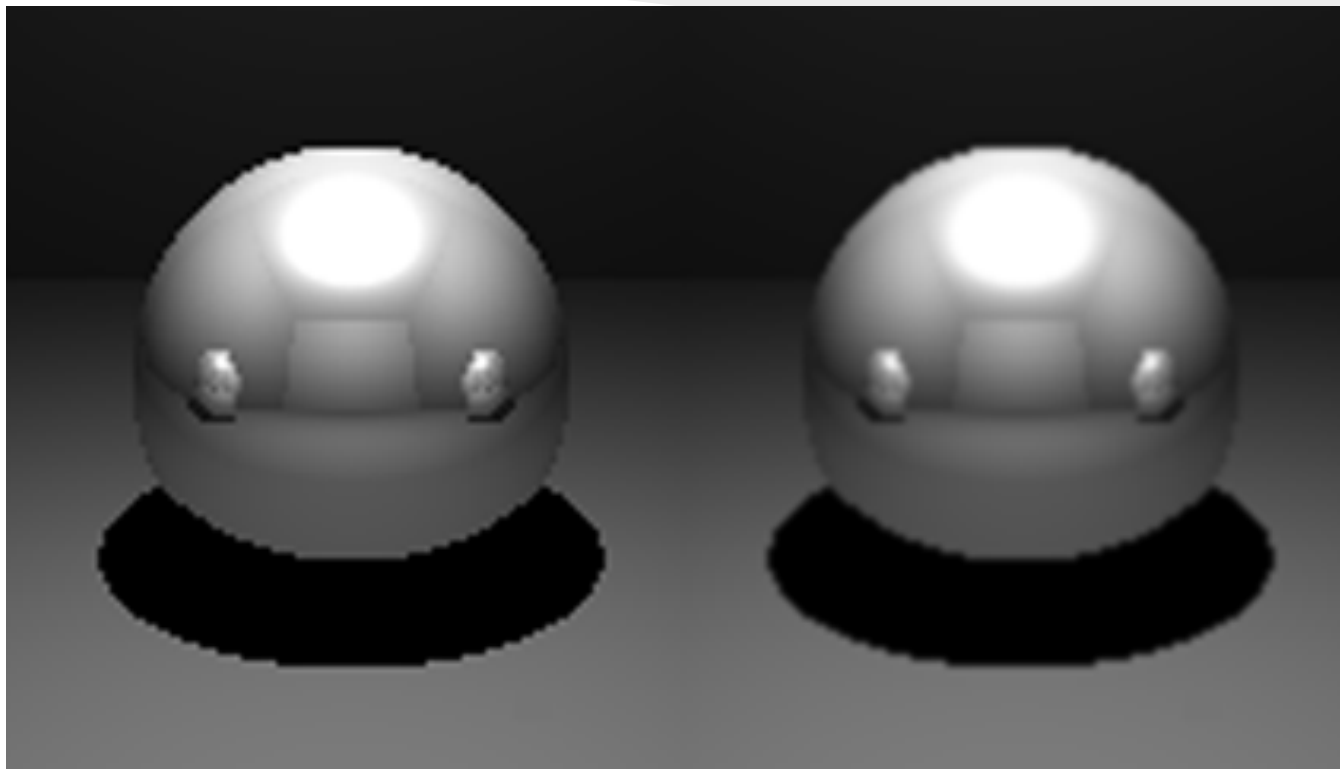
Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.



Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

```
for( driver = 0; driver < totalDriverTypes; ++driver )
{
    result = D3D11CreateDeviceAndSwapChain( 0, driverTypes[driver], 0,
        creationFlags, featureLevels, totalFeatureLevels,
        D3D11_SDK_VERSION, &swapChainDesc, &swapChain_,
        &d3dDevice_, &featureLevel_, &d3dContext_ );

    if( SUCCEEDED( result ) ) {
        driverType_ = driverTypes[driver];
        break;
    }
}
if( FAILED( result ) )
{
    DXTRACE_MSG( "Failed to create the Direct3D device!" );
    return false;
}
```

Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

A render target view is a Direct3D resource written to by the output merger stage. In order for the output merger to render to the swap chain's back buffer (secondary buffer), we create a render target view of it.

The primary and secondary rendering buffers of the swap chain are color textures, and to obtain a pointer to it we call the swap chain function `GetBuffer`.

Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

```
HRESULT result = swapChain_ ->GetBuffer( 0, __uuidof( ID3D11Texture2D ),
                                           ( LPVOID* )&backBufferTexture );
if( FAILED( result ) )
{
    DXTRACE_MSG( "Failed to get the swap chain back buffer!" );
    return false;
}
result = d3dDevice_ ->CreateRenderTargetView( backBufferTexture, 0,
                                              &backBufferTarget_ );
if( backBufferTexture )
    backBufferTexture ->Release( );
```

Initializing DirectX

1. Define the device types and feature levels.

2. Create device, rendering context, and swap chain.

3. Create the render target.

4. Set the viewport.

Setting the render target description parameter to null gives us a view of the entire surface at mip level 0.

Each time we want to render to a specific render target, we must set it first before any drawing calls. This is done by calling `OMSetRenderTarget`, which is a function that is part of the output merger (hence the OM in `OMSetRenderTarget`).

The `OMSetRenderTarget` function takes as parameters the number of views we are binding in this function call, the list of render target views, and the depth/stencil views.

```
d3dContext_ ->OMSetRenderTargets( 1, &backBufferTarget_, 0 );
```

Initializing DirectX

1. Define the device types and feature levels.
2. Create device, rendering context, and swap chain.
3. Create the render target.
4. Set the viewport.

The viewport defines the area of the screen we are rendering to.

For split-screen games we can create two viewports, with one defining the upper portion of the screen and one for the lower portion.

Initializing DirectX

1. Define the device types and feature levels.
2. Create device, rendering context, and swap chain.
3. Create the render target.
4. Set the viewport.

```
D3D11_VIEWPORT viewport;  
viewport.Width = static_cast<float>(width);  
viewport.Height = static_cast<float>(height);  
viewport.MinDepth = 0.0f;  
viewport.MaxDepth = 1.0f;  
viewport.TopLeftX = 0.0f;  
viewport.TopLeftY = 0.0f;
```

```
d3dContext_ ->RSSetViewports( 1, &viewport );
```


DirectX Rendering

Rendering to the screen takes place in a few different steps. The first step is usually to **clear** any necessary render target surfaces.

It is not necessary to clear the color buffer before rendering in most commercial games because the sky and environment geometry ensures every pixel will be overridden in the color buffer anyway, making the clearing step unnecessary.

To clear the screen, we are specifying the color as the color we want the background shaded to. This color is a red, green, blue, and alpha array with color values specified in the 0.0 to 1.0 range. In this case 0.0 is nonintensity, and 1.0 is full intensity.

DirectX Rendering

The next step is to draw the scene's geometry.

The last step is to display the rendered buffer to the screen by calling the swap chain's Present function.

```
float clearColor[4] = { 0.0f, 0.0f, 0.25f, 1.0f };
```

```
d3dContext_ ->ClearRenderTargetView( backBufferTarget_, clearColor );
```

```
swapChain_ ->Present( 0, 0 );
```

DirectX Clean up

The final thing to do in any Direct3D application is to clean up and release the objects that you've created.

COM objects keep a reference count that tells the system when it's safe to remove objects from memory. By using the Release function, you decrement the reference count for an object. When the reference count reaches 0, the system reclaims these resources.

```
if( backBufferTarget_ ) backBufferTarget_->Release( );  
if( swapChain_ ) swapChain_->Release( );  
if( d3dContext_ ) d3dContext_->Release( );  
if( d3dDevice_ ) d3dDevice_->Release( );
```

Test 2: First DirectX project

goo.gl/forms/mwipx2CWTl