How to walk on a top of a nuclear reactor not leaving lovely Excel

Okay. One morning we woke up on an uninhabited island.

We have nothing but a laptop with 64-bit Windows 10 and Excel 2016. Don’t worry, that’ll be enough to build any 3D model in any detail you may need. Here I'll show you how.

The main idea is to use common WinAPI libraries. From this point, it doesn’t matter which language to use for calling API functions. Excel 2016 offers VBA. Not bad!

To create a 3D environment we’ll use OpenGL as well.

This is all we need to know before the start.

# Part 1

First off, we have to write the Hello World framework.

Launch the Excel App, create a new Excel book, and press Alt+F11 to open the VBA. In the VBA window menu, we choose “Insert” -> “Module”. Now we have a white field with a lonely text: “Option Explicit”. This will be our workspace.

To show the “Hello World” greeting, we can use the built-in VBA function: MsgBox ("Hello World") but our primary goal is to explore how WinAPI works in the VBA environment so we use the "User32" library instead.

Yes, the "User32" and "Kernel32", as well as many other libraries, are accessible from VBA!

Maybe that’s the reason why some people consider VBA unsafe.

Here’s how to declare the API functions in VBA:

Declare PtrSafe Function MessageBoxW Lib "User32" (ByVal hWnd As LongPtr, ByVal lpText As LongPtr, ByVal lpCaption As LongPtr, ByVal uType As LongPtr) As LongPtr

We must use the PtrSafe keyword in combination with the LongPtr data type to provide compatibility with 32-bit and 64-bit platforms.

All arguments must be declared ByVal (by value) and must have the same size: 4 bytes for 32 bits and 8 bytes for 64 bits, respectively.

The following table helps understanding the principles of data size correspondence.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WinAPI data type | Example | Size in memory | VBA variable type | Notes |
| Arguments passed to functions | ByVal msg As LongPtr | 4 bytes for 32 bits  8 bytes for 64 bits | Long for 32 bits  LongLong for 64 bits  LongPtr for both |  |
| Handles, pointers | hWnd, hMenu,  lpText, lpWndClass | 4 bytes for 32 bits  8 bytes for 64 bits | Long for 32 bits  LongLong for 64 bits  LongPtr for both |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

We also must use the Unicode versions of the functions. MessageBoxW stands for Unicode and MessageBoxA is for ANSI. The difference is that Unicode takes 2 bytes per symbol while ANSI takes only 1 byte. However, VBA works correctly only with the 2-byte symbol representation.

As we can see, three of arguments are addresses in memory: hWnd, lpText, and lpCaption. The fourth argument is a constant.

There is no Internet on our uninhabited island so we can’t download header files such as "windows.h".

So we need to declare certain constants like this:

Public Const MB\_OK = 0

Public Const MB\_ICONINFORMATION = &H40

Our message box will contain only one button and it will show the info mark.

After all declarations, we can call the function.

According to VBA syntax, executable code must be contained in a procedure:

Sub HelloWorldMessageProc()

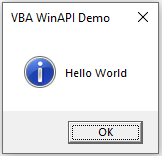
    Call MessageBoxW(0, StrPtr("Hello World"), StrPtr("VBA WinAPI Demo"), MB\_OK Or MB\_ICONINFORMATION)

End Sub

Since there is no main window yet, hWnd = 0.

The StrPtr function returns the addresses of text strings in memory. VBA’s environment has the special containment for text strings. So don’t confuse it with the VarPtr function that is used for getting addresses of other types of variables.

If everything has been done right then by pressing F5 we can get a little message box named "VBA WinAPI Demo" with a message "Hello World":



Congrats! We’ve just proven that VBA can call WinAPI functions.

Now we can use the full range of this powerful toolset.

Part 2

The next crazy step is to create the separate window application inside the launched Excel and VBA.

Why not? Come on!

We’ll register the main window and create the main loop to get window messages, and create a window procedure to proceed them.

Don’t you believe? Just take a look.

Here’s the full list of needed WinAPI functions:

'WinMain

Declare PtrSafe Function GetModuleHandleW Lib "Kernel32" (ByVal lpModuleName As LongPtr) As LongPtr

Declare PtrSafe Function LoadIconW Lib "User32" (ByVal hInstance As LongPtr, ByVal lpIconName As LongPtr) As LongPtr

Declare PtrSafe Function LoadCursorW Lib "User32" (ByVal hInstance As LongPtr, ByVal lpCursorName As LongPtr) As LongPtr

Declare PtrSafe Function GetStockObject Lib "Gdi32" (ByVal i As LongPtr) As LongPtr

Declare PtrSafe Function RegisterClassExW Lib "User32" (ByVal lpWndClass As LongPtr) As LongPtr

Declare PtrSafe Function CreateWindowExW Lib "User32" (ByVal dwExStyle As LongPtr, ByVal lpClassName As LongPtr, ByVal lpWindowName As LongPtr, ByVal dwStyle As LongPtr, ByVal x As LongPtr, ByVal y As LongPtr, ByVal nWidth As LongPtr, ByVal nHeight As LongPtr, ByVal hWndParent As LongPtr, ByVal hMenu As LongPtr, ByVal hInstance As LongPtr, ByVal lpParam As LongPtr) As LongPtr

Declare PtrSafe Function ShowWindow Lib "User32" (ByVal hWnd As LongPtr, ByVal nCmdShow As LongPtr) As LongPtr

Declare PtrSafe Function UpdateWindow Lib "User32" (ByVal hWnd As LongPtr) As LongPtr

Declare PtrSafe Function PeekMessageW Lib "User32" (ByVal lpMsg As LongPtr, ByVal hWnd As LongPtr, ByVal wMsgFilterMin As LongPtr, ByVal wMsgFilterMax As LongPtr, ByVal wRemoveMsg As LongPtr) As LongPtr

Declare PtrSafe Function TranslateMessage Lib "User32" (ByVal lpMsg As LongPtr) As LongPtr

Declare PtrSafe Function DispatchMessageW Lib "User32" (ByVal lpMsg As LongPtr) As LongPtr

Declare PtrSafe Function GetLastError Lib "Kernel32" () As LongPtr

'WndProc

Declare PtrSafe Function MessageBoxW Lib "User32" (ByVal hWnd As LongPtr, ByVal lpText As LongPtr, ByVal lpCaption As LongPtr, ByVal uType As LongPtr) As LongPtr

Declare PtrSafe Function DefWindowProcW Lib "User32" (ByVal hWnd As LongPtr, ByVal msg As LongPtr, ByVal wParam As LongPtr, ByVal lParam As LongPtr) As LongPtr

Declare PtrSafe Function DestroyWindow Lib "User32" (ByVal hWnd As LongPtr) As LongPtr

Declare PtrSafe Function PostQuitMessage Lib "User32" (ByVal nExitCode As LongPtr) As LongPtr

Structures in VBA may be created as user-defined variable types:

'Structures

Type WNDCLASSEX

cbSize As Long

style As Long

lpfnWndProc As LongPtr

cbClsExtra As Long

cbWndExtra As Long

hInstance As LongPtr

hIcon As LongPtr

hCursor As LongPtr

hbrBackground As LongPtr

lpszMenuName As LongPtr

lpszClassName As LongPtr

hIconSm As LongPtr

End Type

Public wcx As WNDCLASSEX

Public lpWndClass As LongPtr 'Pointer

Type POINT2D

X As Long

Y As Long

End Type

Public pt As POINT2D

Public lpMsgPt As LongPtr

Type Msg

hWnd As LongPtr

message As Long

wParam As LongPtr

lParam As LongPtr

time As Long

pt As POINT2D

lPrivate As LongPtr

End Type

Public wmsg As Msg

Public lpMsg As LongPtr 'Pointer

Note that 32-bit DWORD variables are declared in VBA As Long. Do not confuse them with HINSTANCE, HICON, HCURSOR and other handles and pointers, which are actually addresses in memory that can be 32-bit or 64-bit depending on the system and thus must be declared As LongPtr.

The constants that usually can be found in ‘windows.h’ and other header files we have to type manually:

'Const

Public Const PM\_REMOVE = 1

'Window Styles

Public Const CS\_VREDRAW = 1

Public Const CS\_HREDRAW = 2

Public Const IDI\_APPLICATION = 32512

Public Const IDC\_ARROW = 32512

Public Const CW\_USEDEFAULT = &H80000000

Public Const WS\_OVERLAPPED = 0

Public Const WS\_CAPTION = &HC00000

Public Const WS\_SYSMENU = &H80000

Public Const WS\_THICKFRAME = &H40000

Public Const WS\_MINIMIZEBOX = &H20000

Public Const WS\_MAXIMIZEBOX = &H10000

Public Const WS\_OVERLAPPEDWINDOW = WS\_OVERLAPPED Or WS\_CAPTION Or WS\_SYSMENU Or WS\_THICKFRAME Or WS\_MINIMIZEBOX Or WS\_MAXIMIZEBOX

Public Const SW\_SHOW = 5

'Messages

Public Const WM\_DESTROY = 2

Public Const WM\_CLOSE = &H10

Public Const WM\_QUIT = &H12

'Virtual Keys

Public Const MB\_YESNO = 4

Public Const MB\_ICONQUESTION = &H20

Public Const IDYES = 6

'Brushes

Public Const GRAY\_BRUSH = 2

'Custom Values

'Screen

Public Const DEFAULT\_SCREEN\_WIDTH = 1024

Public Const DEFAULT\_SCREEN\_HEIGHT = 768

We’ll need to store a couple of handles in global variables:

'Global Handles

Public ghInst As LongPtr

Public ghWnd As LongPtr

In the Part 1, we wrote the text strings and found their handles immediately in the call operation:

Call MessageBoxW(0, StrPtr("Hello World"), StrPtr("VBA WinAPI Demo"), MB\_OK Or MB\_ICONINFORMATION)

In the Part 2, we’ll see the harder way. We’ll use global variables for our text strings and their handles.

'Strings

Public szMsgText As String

Public lpszMsgText As Long

Public szMsgTitle As String

Public lpszMsgTitle As Long

Let’s add one global variable for the debug purpose:

'Debug

Public nLastError As Long

Now we are ready to write the WinMain function:

Function WinMain(ByVal hInstance As LongPtr, ByVal hPrevInstance As LongPtr, ByVal lpCmdLine As LongPtr, ByVal nCmdShow As LongPtr)

'Text

Dim szCaption As String

Dim lpszCaption As Long

'Main Cycle

Dim nPeek As Long

'Debug

Dim nWndClass As Long 'For Debug Purpose

'Pointers

lpWndClass = VarPtr(wcx.cbSize)

lpMsg = VarPtr(wmsg.hWnd)

wcx.cbSize = Len(wcx)

wcx.style = CS\_HREDRAW Or CS\_VREDRAW

wcx.lpfnWndProc = GetAddr(AddressOf WndProc)

wcx.cbClsExtra = 0

wcx.cbWndExtra = 0

wcx.hInstance = GetModuleHandleW(0)

If wcx.hInstance = 0 Then

Call MsgBox("GetModuleHandle Error: " & GetLastError())

Exit Function

Else

ghInst = wcx.hInstance

End If

wcx.hIcon = LoadIconW(0, IDI\_APPLICATION)

wcx.hCursor = LoadCursorW(0, IDC\_ARROW)

wcx.hbrBackground = GetStockObject(GRAY\_BRUSH)

wcx.lpszMenuName = 0

wcx.lpszClassName = StrPtr("MainWindowClassName")

wcx.hIconSm = wcx.hIcon

nWndClass = RegisterClassExW(lpWndClass)

If nWndClass = 0 Then

nLastError = GetLastError()

If nLastError = 1410 Then

'Call MsgBox("RegisterClass Error 1410 (0x582): ERROR\_CLASS\_ALREADY\_EXISTS")

Else

Call MsgBox("RegisterClass Error: " & nLastError)

Exit Function

End If

End If

szCaption = "VBA WinAPI OpenGL Demo"

lpszCaption = StrPtr(szCaption)

ghWnd = CreateWindowExW(0, wcx.lpszClassName, lpszCaption, WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, DEFAULT\_SCREEN\_WIDTH, DEFAULT\_SCREEN\_HEIGHT, 0, 0, wcx.hInstance, 0)

If ghWnd = 0 Then

Call MsgBox("CreateWindow Error: " & GetLastError())

Exit Function

End If

Call InitializeGL

Call ShowWindow(ghWnd, SW\_SHOW)

Call UpdateWindow(ghWnd)

WinMainLoop:

nPeek = PeekMessageW(lpMsg, 0, 0, 0, PM\_REMOVE)

If nPeek = 0 Then

Call DrawGLScene 'We'll draw the OpenGL Scene outside the WndProc

Else

If wmsg.message = WM\_QUIT Then

WinMain = wmsg.wParam

Exit Function 'Don't use Call ExitProcess(0)

Else

Call TranslateMessage(lpMsg)

Call DispatchMessageW(lpMsg)

End If

End If

GoTo WinMainLoop

End Function

The WinMain function in VBA looks familiar but there is one little trick.

To register the main window class, we need to load the handle (i.e. address) of WndProc to the wcx structure. But for some reason, the VBA’s inner AddressOf operator cannot be used in assignment operations immediately.

<https://citforum.ru/programming/vb/vba_winapi/7.shtml>

The decision is to pass the WndProc’s address to an extra procedure that will return it back:

Function WinMain(...)

    ...

    wcx.lpfnWndProc = GetAddr(AddressOf WndProc)

    ...

End Function

Function GetAddr(ByVal lpProc As Long) As Long

    GetAddr = lpProc

End Function

At this step, functions InitializeGL() and DrawGLScene() are just dummies that we’ll fill in the Part 3:

Function InitializeGL()

End Function

Function DrawGLScene()

End Function

The code of WndProc also is not surprising:

Function WndProc(ByVal hWnd As Long, ByVal message As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Select Case message

Case WM\_CLOSE

GoTo lbl\_wmClose

Case WM\_DESTROY

GoTo lbl\_wmDestroy

End Select

lbl\_DefWndProc:

WndProc = DefWindowProcW(hWnd, message, wParam, lParam)

Exit Function

lbl\_wmClose:

szMsgText = "Close?"

lpszMsgText = StrPtr(szMsgText)

szMsgTitle = "Such A Good Application"

lpszMsgTitle = StrPtr(szMsgTitle)

If MessageBoxW(hWnd, lpszMsgText, lpszMsgTitle, MB\_YESNO Or MB\_ICONQUESTION) = IDYES Then

Call DestroyWindow(hWnd)

End If

GoTo lbl\_WndProc\_Return0

lbl\_wmDestroy:

Call PostQuitMessage(0)

lbl\_WndProc\_Return0:

WndProc = 0

End Function

The entry point to our “application” will look like this:

Sub Start()

Call WinMain(0, 0, 0, 0)

End Sub

For our convenience, we’ll create in the blank Excel sheet a button that will launch our 3D world. Here’s VBA code for this:

Public Sub ShowButton()

Application.ScreenUpdating = False

ActiveSheet.Buttons.Delete

Dim btnStart As Button

Dim t As Range

Set t = ActiveSheet.Cells(2, 2)

Set btnStart = ActiveSheet.Buttons.Add(t.Left, t.Top, t.Width, t.Height)

With btnStart

.OnAction = "Start"

.Caption = "A3-5"

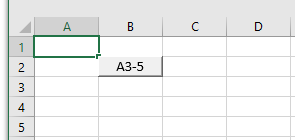
.Name = "Start"

End With

Application.ScreenUpdating = True

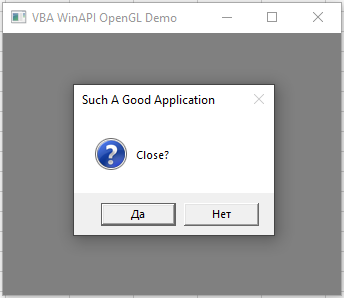
End Sub

Paste all the code above, place the cursor somewhere inside the Public Sub ShowButton(), and press F5. You’ll see the new button:



How we have everything that we need to begin a new life inside Excel.

Hit the magic button!



It’s wonderful, isn’t it?

A real window application that we’ve built with bare Excel. It can receive and proceed messages and it acts like a separate program. Exciting! Fascinating!

Part 3

Go graphics!

To ignite OpenGL we’ll add these functions:

'GDI & Paint

Declare PtrSafe Function GetDC Lib "User32" (ByVal hdc As Long) As Long

Declare PtrSafe Function ReleaseDC Lib "User32" (ByVal hWnd As Long, ByVal hdc As Long) As Long

Declare PtrSafe Function GetClientRect Lib "User32" (ByVal hWnd As Long, ByVal lpRect As Long) As Long

'OpenGL

Declare PtrSafe Function ChoosePixelFormat Lib "Gdi32" (ByVal hdc As Long, ByVal ppfd As Long) As Long

Declare PtrSafe Function SetPixelFormat Lib "Gdi32" (ByVal hdc As Long, ByVal format As Long, ByVal ppfd As Long) As Long

Declare PtrSafe Function SwapBuffers Lib "Gdi32" (ByVal hdc As Long) As Long

Declare PtrSafe Function wglCreateContext Lib "Opengl32" (ByVal hdc As Long) As Long

Declare PtrSafe Function wglMakeCurrent Lib "Opengl32" (ByVal hdc As Long, ByVal hGLRC As Long) As Long

Declare PtrSafe Function wglDeleteContext Lib "Opengl32" (ByVal hGLRC As Long) As Long

Declare PtrSafe Function glViewport Lib "Opengl32" (ByVal x As Long, ByVal y As Long, ByVal width As Long, ByVal height As Long) As Long

Declare PtrSafe Function glMatrixMode Lib "Opengl32" (ByVal mode As Long) As Long

Declare PtrSafe Function glLoadIdentity Lib "Opengl32" () As Long

Declare PtrSafe Function gluPerspective Lib "Glu32" (ByVal fovy As Double, ByVal aspect As Double, ByVal zNear As Double, ByVal zFar As Double) As Long

Declare PtrSafe Function gluLookAt Lib "Glu32" (ByVal eyex As Double, ByVal eyey As Double, ByVal eyez As Double, ByVal centerx As Double, ByVal centery As Double, ByVal centerz As Double, ByVal upx As Double, ByVal upy As Double, ByVal upz As Double) As Long

Declare PtrSafe Function glClearColor Lib "Opengl32" (ByVal red As Single, ByVal green As Single, ByVal blue As Single, ByVal alpha As Single) As Long

Declare PtrSafe Function glClear Lib "Opengl32" (ByVal mask As Long) As Long

Declare PtrSafe Function glBegin Lib "Opengl32" (ByVal mode As Long) As Long

Declare PtrSafe Function glEnd Lib "Opengl32" () As Long

Declare PtrSafe Function glColor3f Lib "Opengl32" (ByVal red As Single, ByVal green As Single, ByVal blue As Single) As Long

Declare PtrSafe Function glVertex3f Lib "Opengl32" (ByVal x As Single, ByVal y As Single, ByVal z As Single) As Long

Three new structures:

Type RECT

left As Long 'Stack Top

top As Long

right As Long

bottom As Long 'Stack Bottom

End Type

Public RectMain As RECT

Public lpRectMain As Long 'Pointer

Type PAINTSTRUCT

hdc As Long 'Stack Top

fErase As Long

rcPaint As RECT

fRestore As Long

fIncUpdate As Long

rgbReserved(31) As Byte 'Stack Bottom

End Type

Public ps As PAINTSTRUCT

Public lpPS As Long 'Pointer

Type PIXELFORMATDESCRIPTOR

nSize As Integer 'Stack Top

nVersion As Integer

dwFlags As Long

iPixelType As Byte

cColorBits As Byte

cRedBits As Byte

cRedShift As Byte

cGreenBits As Byte

cGreenShift As Byte

cBlueBits As Byte

cBlueShift As Byte

cAlphaBits As Byte

cAlphaShift As Byte

cAccumBits As Byte

cAccumRedBits As Byte

cAccumGreenBits As Byte

cAccumBlueBits As Byte

cAccumAlphaBits As Byte

cDepthBits As Byte

cStencilBits As Byte

cAuxBuffers As Byte

iLayerType As Byte

bReserved As Byte

dwLayerMask As Long

dwVisibleMask As Long

dwDamageMask As Long 'Stack Bottom

End Type

Public pfd As PIXELFORMATDESCRIPTOR

Public lpPFD As Long 'Pointer

Let’s add the WM\_SIZE message to rebuild the viewport correctly if the window is maximized or resized.

Public Const WM\_SIZE = 5

and a few specific constants for OpenGL:

Public Const PFD\_SUPPORT\_OPENGL = &H20

Public Const PFD\_DOUBLEBUFFER = 1

Public Const PFD\_DRAW\_TO\_WINDOW = 4

Public Const PFD\_TYPE\_RGBA = 0

Public Const PFD\_MAIN\_PLANE = 0

Public Const GL\_MODELVIEW = &H1700

Public Const GL\_PROJECTION = &H1701

Public Const GL\_COLOR\_BUFFER\_BIT = &H4000

Public Const GL\_DEPTH\_BUFFER\_BIT = &H100

Public Const GL\_LINES = 1

Public Const GL\_TRIANGLES = 4

Public Const GL\_TRIANGLE\_STRIP = 5

Public Const GL\_QUADS = 7

New global variables:

Public ghDC As LongPtr

Public ghRC As LongPtr

Public iPixelFormat As LongPtr

WinMain doesn’t change.

In WndProc we must add some new code:

'Pointers

lpRectMain = VarPtr(RectMain.left)

lpPS = VarPtr(ps.hdc)

A new kind of message:

Case WM\_SIZE

GoTo lbl\_wmSize

Proceed it:

lbl\_wmSize:

Call GetClientRect(hWnd, lpRectMain)

Call GLResize(CSng(RectMain.right - RectMain.left), CSng(RectMain.bottom - RectMain.top))

GoTo lbl\_WndProc\_Return0

Add some routine to close the “Application” properly:

lbl\_wmClose:

szMsgText = "Close?"

lpszMsgText = StrPtr(szMsgText)

szMsgTitle = "Such A Good Application"

lpszMsgTitle = StrPtr(szMsgTitle)

If MessageBoxW(hWnd, lpszMsgText, lpszMsgTitle, MB\_YESNO Or MB\_ICONQUESTION) = IDYES Then

Call wglMakeCurrent(0, 0)

Call wglDeleteContext(ghRC)

Call ReleaseDC(hWnd, ghDC)

Call DestroyWindow(hWnd)

End If

GoTo lbl\_WndProc\_Return0

Fill the Dummies that we left in the Part 2:

Function InitializeGL()

'PIXELFORMATDESCRIPTOR pfd

pfd.nSize = 40 'sizeof( PIXELFORMATDESCRIPTOR )

pfd.nVersion = 1 'always 1

pfd.dwFlags = PFD\_SUPPORT\_OPENGL Or PFD\_DOUBLEBUFFER Or PFD\_DRAW\_TO\_WINDOW

pfd.iPixelType = PFD\_TYPE\_RGBA

pfd.cColorBits = 24

pfd.cRedBits = 0

pfd.cRedShift = 0

pfd.cGreenBits = 0

pfd.cGreenShift = 0

pfd.cBlueBits = 0

pfd.cBlueShift = 0

pfd.cAlphaBits = 0

pfd.cAlphaShift = 0

pfd.cAccumBits = 0

pfd.cAccumRedBits = 0

pfd.cAccumGreenBits = 0

pfd.cAccumBlueBits = 0

pfd.cAccumAlphaBits = 0

pfd.cDepthBits = 32

pfd.cStencilBits = 0

pfd.cAuxBuffers = 0

pfd.iLayerType = PFD\_MAIN\_PLANE

pfd.bReserved = 0

pfd.dwLayerMask = 0

pfd.dwVisibleMask = 0

pfd.dwDamageMask = 0

lpPFD = VarPtr(pfd.nSize)

ghDC = GetDC(ghWnd)

iPixelFormat = ChoosePixelFormat(ghDC, lpPFD)

If iPixelFormat = 0 Then

MsgBox ("ChoosePixelFormat Error: " & GetLastError())

End If

Call SetPixelFormat(ghDC, iPixelFormat, lpPFD)

ghRC = wglCreateContext(ghDC)

If ghRC = 0 Then

nLastError = GetLastError()

If nLastError = 2000 Then

MsgBox ("wglCreateContext Error 2000 (0x7D0): ERROR\_INVALID\_PIXEL\_FORMAT")

Else

MsgBox ("wglCreateContext Error: " & nLastError)

End If

End If

Call wglMakeCurrent(ghDC, ghRC)

End Function

Draw our “Hello triangle”

Function DrawGLScene()

'Call glClearColor(0.75, 0.75, 0.75, 0)

'Call glClear(GL\_COLOR\_BUFFER\_BIT Or GL\_DEPTH\_BUFFER\_BIT)

Call glBegin(GL\_TRIANGLES)

Call glColor3f(1, 0, 0) 'Red

Call glVertex3f(-0.5, -0.5, 0)

Call glColor3f(0, 1, 0) 'Green

Call glVertex3f(0, 0.5, 0)

Call glColor3f(0, 0, 1) 'Blue

Call glVertex3f(0.5, -0.5, 0)

Call glEnd

Call SwapBuffers(ghDC)

End Function

And make it resizeable:

Function GLResize(ByVal width As Single, ByVal height As Single)

Dim aspect As Single

aspect = width / height

Call glViewport(0, 0, width, height)

Call glMatrixMode(GL\_PROJECTION)

Call glLoadIdentity

Call gluPerspective(45#, aspect, 1, 1)

Call glMatrixMode(GL\_MODELVIEW)

Call glLoadIdentity

Call gluLookAt(0.125, 0.125, 0.125, 0, 0, 0, 0, 1, 0)

End Function