DirectX in C#

Tomáš SMLSAL
University of West Bohemia
Univerzitní 8, BOX 314
306 14 Plzeň, Czech Republic
tscz@centrum.cz

Václav SKALA
University of West Bohemia
Univerzitní 8, BOX 314
306 14 Plzeň, Czech Republic
skala@kiv.zcu.cz

ABSTRACT

Several techniques of implementing DirectX functionality into C# application will be presented. Their common attribute is an idea of component object model (COM) because DirectX is based on component technology. This paper will be focused on use of DirectX graphical capabilities within the C# code of the Microsoft .NET Framework. Three main techniques will be described. First, COM interoperability which allows us to decide what specific functionality to use. There will be also mentioned some basic principles like memory management and garbage collector (GC) and some approaches based on wrapper classes. Second, Visual Basic type library that includes all the functionality, and third, the complete solution known as DirectX 9.0. Each technique will be supported with a code snippet and with several reasons stating its suitability. Nowadays, the need for security can be more important than for the efficiency. This also gives a right answer for the question of how good is solution provided by use of DirectX and .NET Framework.

Keywords

Computer graphics, DirectX, Direct3D, DirectDraw, C#, .NET Framework, COM, implementation, interface, wrapper class, security, efficiency, Microsoft.

1. INTRODUCTION

The purpose of this paper is to provide a basic idea about implementation of DirectX graphical interface in the code of C# language in the .NET Framework. A goal is to have such an environment where the code for C# looks similar to the C++ one. This work is a part of project ROTOR, more detailed information is placed at [Her03]. DirectX version 8.1 and above is assumed, if not said explicitly. Complete information on DirectX and .NET Framework can be found in electronic resources [MSDN02].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1st Int. Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing

February 6-8, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press
ISBN 80-903100-3-6

.NET Framework

The .NET Framework Environment where the C# code is executed. It uses Garbage Collector (GC) for system memory management, which automates such tasks as a memory allocation, release, fragmentation, etc. No pointers are allowed here (managed code) but if necessary, the unmanaged code can be entered. In this mode pointers are allowed but GC doesn't work - it is a developer responsibility to manage the memory. Therefore the managed code should be used but there is a problem: DirectX is using pointer parameters to pass data into its functions. How to face this pointer-problem will be shown later in the paragraph “COM Benefits”.

What is it a Direct3D?

DirectX was originally prepared for game and multimedia developers and it is a set of application interfaces (APIs). It provides a low-level access to hardware functionality of available peripheral hardware devices like a graphical adapter, sound card and so on. Very important fact is that all this technology is based on a Component Object Model

Project supported by the Ministry of Education of The Czech Republic - Project MSM 235200005 and Microsoft Research Ltd. (UK).

(COM). In other words, DirectX is a set of COM components, each providing some interfaces which can be divided into subsets with a similar functionality. One of the subsets does all about the graphics and is called DirectX Graphics. It combines previous 3D and 2D graphic components (Direct3D and DirectDraw) into one and the name Direct3D remained for both. (Now, the entire planar graphic must be done via 3D component.)

COM benefits

The runtime of .NET Framework has some features like memory management based on garbage collector (GC). It automatically controls the lifetime of existing objects, their location in a memory to prevent fragmentation and removes them from memory since there is no reference to them. A code written for this managed environment can be called safe code and no pointers are allowed. Having a reference to an object, GC can shift the object in memory and the reference is still referencing it. But once the pointer is initialized to some address, GC must keep away from the object lying there to avoid its possible shifting and invalidating the pointer. To switch to this unmanaged mode, where pointers are used, the unsafe code has to be used.

To pass data into DirectX methods, pointers should be necessary as well as the unmanaged mode. But the managed one is the preferred one.

Wrapping task can be defined as a process when migrating some functionality from foreign development environment into ours without changes at the origin source code. In other words, it can be also named as porting as in [Han03]. To create a port of some dynamically linked library (.dll) means to somehow provide headers of all necessary functions and to do all the necessary steps for the .dll import [Han03]. But having the original functionality in a COM, it is simple to let the .NET Framework runtime to do everything automatically. The runtime has methods for handling components written in an unmanaged

The advantage that DirectX is a COM based is highly welcome. The .NET Framework runtime environment can save a lot of work to developer in a wrapping task because of its runtime callable wrappers feature. The functionality of GC can be used although the pointers are needed as well. Each time the method of a COM is called, the runtime callable wrapper (RCW) is automatically created for accessing the unmanaged code of that COM. It is created every time that the call occurs. This could seem to be unacceptably high overhead cost, but, if considering the fact that for e.g. rendering 10 or 10 billions facets takes only one call and one RCW build, it is feasible.

All that developer has to do is a COM interfaces registration. Therefore the problem of wrapping is not as difficult as in [Han03] and it is not necessary to deal with some specific problems. In the following paragraphs, important procedures of how to do it will be described.

2. COM COCLASSES

This is the first general approach that uses the COM interoperability. Essentials about COM interoperability can be found in [Vis03]. At first, a list of all component interfaces is taken and for each interface is done registration as in the Figure 1. example for an `IDirect3D`.

```
using System;
using System.Runtime.InteropServices;

namespace Sample
{
    [ComImport, Guid("3BBA0080-2421-11CF-A31A-00AA00B93356")]
    class IDirect3D {}

    class Test
    {
        static void oldMain()
        {
            IDirect3D iDirect3D = new IDirect3D();
            // Creates a COM object wrapper
        }
    }
```

Figure 1: COM registration.

Immediately after this declaration, the COM object is ready for initialization and use.

Instantiating an `IDirect3D` instance causes a corresponding COM instantiation and all its methods are called via the reference `iDirect3D`. The list of necessary GUIDs (Globally Unique IDentifiers) can be retrieved from header files of DirectX SDK, downloadable from [MSDN02].

This is the most general way of obtaining DirectX functionality in the sense that only necessary parts of the interface are included.

```
using System;
using DxVBLib;
namespace Sample{
    public class CSample
    {
        private DirectX7            m_objDirectX = new DirectX7();
        private DirectDraw7         m_objDirectDraw;
        private DirectDrawSurface7  m_objPrimarySurface;
        private DirectDrawSurface7  m_objBackBufferSurface;
        private DDSURFACEDESC2      m_objPrimarySurfaceDescription;
        private DDSURFACEDESC2      m_objBackBufferSurfaceDescription;
        ...
        public void InitialiseDirectXFullScreen(int iHandle)
        {
            //create DirectDraw object
            m_objDirectDraw = this.DirectX.DirectDrawCreate("");

            //set co-opperative level for full screen
            m_objDirectDraw.SetCooperativeLevel(
                iHandle,
                CONST_DDSCLFLAGS.DDSCL_FULLSCREEN
                | CONST_DDSCLFLAGS.DDSCL_EXCLUSIVE
                );
            m_objDirectDraw.SetDisplayMode(//set display mode
                640, 480, 32, 0, CONST_DDSDMFLAGS.DDSDM_DEFAULT
                );
            //indicate that the caps and backbuffer count will be defined
            m_objPrimarySurfaceDescription.lFlags =
                CONST_DDSURFACEDESCFLAGS.DDSD_CAPS
                | CONST_DDSURFACEDESCFLAGS.DDSD_BACKBUFFERCOUNT;
            m_objPrimarySurfaceDescription.ddsCaps.lCaps =
                CONST_DDSURFACECAPSFLAGS.DDSCAPS_PRIMARYSURFACE
                | CONST_DDSURFACECAPSFLAGS.DDSCAPS_FLIP
                | CONST_DDSURFACECAPSFLAGS.DDSCAPS_COMPLEX;
            //set backbuffer count to 1
            m_objPrimarySurfaceDescription.lBackBufferCount = 1;
            //create primary surface
            this.PrimarySurface =
                this.DirectDraw.CreateSurface(
                    ref m_objPrimarySurfaceDescription
                    );
            //indicate we will set caps of backbuffer
            m_objBackBufferSurfaceDescription.lFlags =
                CONST_DDSURFACEDESCFLAGS.DDSD_CAPS;
            //set the descriptors caps to backbuffer surface
            m_objBackBufferSurfaceDescription.ddsCaps.lCaps =
                CONST_DDSURFACECAPSFLAGS.DDSCAPS_BACKBUFFER;
            //create backbuffer using primary surface
            this.BackBufferSurface =
                this.PrimarySurface.GetAttachedSurface(
                    ref m_objBackBufferSurfaceDescription.ddsCaps); }
        public void FlipBackBufferAndPrimary()...
        public DirectX7 DirectX {
            set { DirectX = value; }
            get { return m_objDirectX; }
        }
        public DirectDraw7 DirectDraw {
            set { DirectDraw = value; }
            get { return m_objDirectDraw; }
        }
        public DirectDrawSurface7 PrimarySurface...
        public DirectDrawSurface7 BackBufferSurface...
    }
}
```

Figure 2: Sample code snippet demonstrating DX7 surface initialization.

Another significant reason for this strategy can be higher level of freedom while mixing components from several versions. However, it is not possible to combine different versions of one component, but it is possible to mix different components, each one from only one version. If, for some reason, a developer needs graphical capabilities of Direct3D version 9.0 and sound from DirectX 6.0, the instructions given above will help him to solve this task efficiently.

3. TYPE LIBRARY DXVBLIB

Compared to previous approach, this solution gives a complete functionality of DirectX by a single command. All to do here is to add a reference to Visual Basic DirectX Type Library named DxVBLib.dll and since it is done, the whole functionality is available through instantiating the needed objects and their references.

In the following example it is shown on creating a Direct3D8 object that supports enumeration and allows the creation of Direct3DDevice8 objects (Figure 3.).

```
using DxVBLib; // +add the ref. in OPTIONS
namespace Sample
    //...
    public class MyPublicClass
    {
        ...
        private Direct3D8 g_pD3D;
        ...
        g_pD3D = this.DirectX.Direct3DCreate8(
        D3D_SDK_VERSION
            );
        ...
    }
```

Figure 3: snippet for type library use.

Until the version DirectX 9.0 was released, this was the simplest method how to implement DirectX in .NET Framework.

In the next example (Figure 2.) is shown similar use of a type library.

4. DIRECTX9.0 (MANAGED)

In this last section the following significant graphic namespaces will be shortly described: Microsoft . DirectX, Microsoft . DirectX.Direct3D and Microsoft.DirectX.DirectDraw.

The namespace **Microsoft.DirectX** provides utility operations and data storage for DirectX application programming, including exception handling, simple helper methods, and structures used for matrix, clipping plane, quaternion, vector manipulation and so forth. **Microsoft.DirectX.Direct3D** enables to manipulate visual models of 3-D objects and take

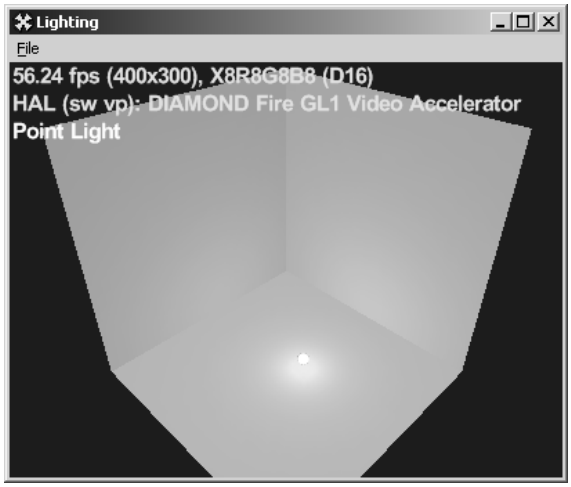


Image 1: Tested sample Lighting.

advantage of hardware acceleration and **Microsoft.DirectX.DirectDraw** that provides functionality across display memory, the hardware blitter, hardware overlay support, and flipping surface support. It seems that small inconsistency appeared because Direct Graphics 8.1 should combine both D3D and DDraw into one, but in the version 9.0 it is formally divided again.

This is the best solution, which provides a complete DirectX functionality in the style of .NET Framework. Code snippets are attached to illustrate the air of object oriented programming with DirectX 9.0 (Figure 4. and 5.).

5. COMPARISON

To provide some information about speed performance, the sample codes of DirectX8.1b in C++ had been compared to DirectX9.0 C# version. The machine configuration was as follows: two Intel Pentium III / 500MHz, 1GB ECC SDRAM, Diamond Fire GL1 Video Accelerator PCI, OS Windows2000, 400x300x32 window mode (See Image 1.).

The method of measurement was done via determining the number of rendered frames per second and from the average for each test was calculated the time in milliseconds with precision provided by number of decimal digits.

See results in Table 1.

C#	C++	Sample type
27,9	23,2	billboarding
10,3	9,4	clipping
15,6	14,0	vertex shader
9,1	6,6	enhanced mesh
17,0	**23,4**	**lights**
7,2	6,3	vertex shader

Table 1: Time [ms] to render the tested scene.

Plotted to Graph 1, it is obvious that some overhead of C# is acceptable with less except of the billboarding and enhanced mesh tests, where the results points to better C++ compiler.

6. CONCLUSION

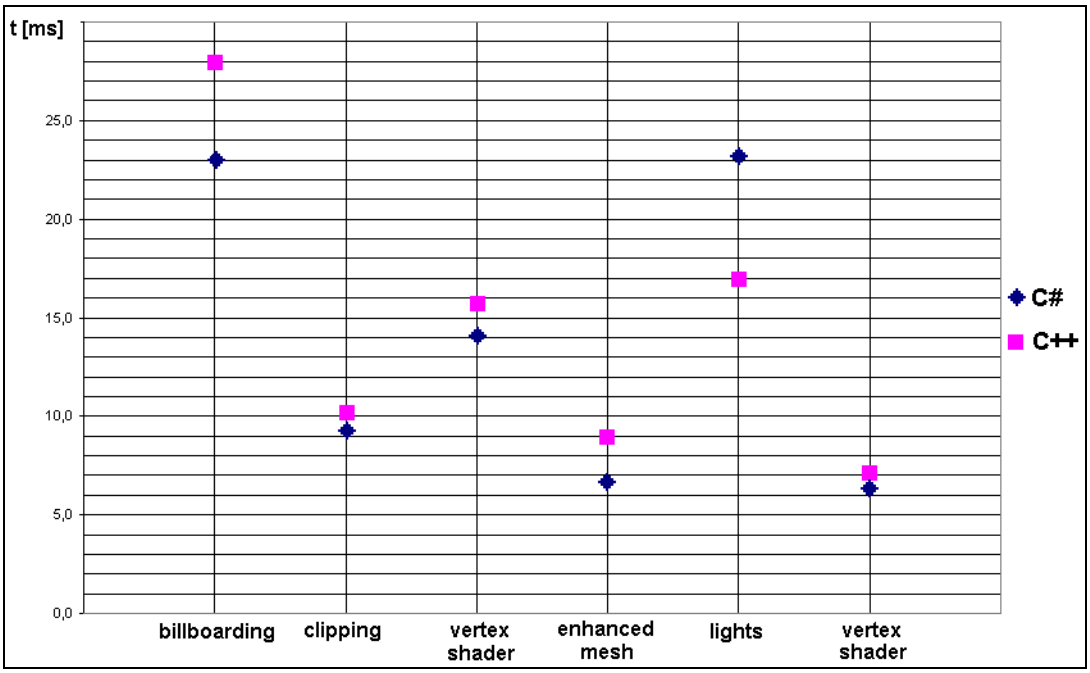
Three methods of DirectX implementation in C# were introduced and described. Until a version 9.0 has been released in December, the only suitable way for C# developers was the second method based on type library import. Since it has been released, the only recommended way is the third one, DirectX9.0 (managed version). With C# and this version can be reached all features of managed runtime .NET Framework environment and OOP even with reasonable overhead compared to C++.

During the tests even some MS SDK samples crashed, some presented error exception and quitted immediately, some deadlocked the machine without any notice and some of them rebooted the machine automatically, so the data were lost several times and it had to be backed up every minute. Maybe there was a problem with SDK installation for Visual C++, which was not installed correctly. Finally, about 60% of samples worked well and it was surprising that DirectX in C# was faster at the lighting test.

7. FUTURE WORK

In the future work, authors of this paper would like to answer the following questions: Was the incorrect installation of MS SDK for Visual C++ the reason why several crashes occurred?

Compared to OpenGL, there exist a lot of online books for academic staff, which are primarily targeted to computer graphics science, and they use a rich OpenGL sample source code support, like e.g. OpenGL Super Bible. Tutorials provided by Microsoft Corp. are targeted more for game developers, thus we will try to find such a resources as mentioned above.



Graph 1: 1: Time [ms] to render scene. Lower means better.

```
using System;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;

namespace MeshSample {
    public class MyMesh {
        protected Material [] MeshMaterials;
        protected Texture [] MeshTextures;
        protected Mesh mesh;
        protected Device d3dDevice;
        protected Matrix matWorld=new Matrix();

        public MyMesh(Device d3device)
        {   d3dDevice=d3device;
            matWorld.Translate(0f,0f,0f);
        }
```

Figure 4: DirectX9.0.

```
using System;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;

namespace MeshSample {
    public class Mesh1 : MyMesh {
        public Mesh1(Device d3device):base(d3device) {
            ExtendedMaterial[] materials=null;
            GraphicsStream o;
            mesh = Mesh.FromFile("mesh.x",MeshFlags.SystemMemory,
                d3dDevice, out o, out materials);
            mesh.OptimizeInPlace(
                MeshFlags.OptimizeCompact
                | MeshFlags.OptimizeAttrSort
                | MeshFlags.OptimizeVertexCache, o);
            //  resizing
            MeshMaterials=new Material[materials.Length];
            MeshTextures=new Texture[materials.Length];

            GraphicsStream g=mesh.LockVertexBuffer(LockFlags.ReadOnly);
            Geometry.ComputeBoundingBox(g,mesh.NumberVertices,
                mesh.VertexFormat, out min,out max);
            mesh.UnlockVertexBuffer();
            g.Close();
            g=null;
            // loop ...
            MeshMaterials[i]=materials[i].Material3D;
            MeshMaterials[i].Ambient = MeshMaterials[i].Diffuse;
            string strTexName = materials[i].TextureFilename;
            if ((strTexName!=null) && (strTexName != ""))
                MeshTextures[i]=
                    TextureLoader.FromFile(d3dDevice, strTexName);
            //end loop ...
            o.Close();
            o=null;
        }
```

Figure 5: DirectX9.0.

8. REFERENCES

[CSC02] **C# Corner**.
http://www.c-sharpcorner.com/Directx.asp

[Han03] Hanák, I., Frank, M., Skala, V.: **OpenGL and VTK interface for .NET**. Accepted for publication in *C# and .NET Technologies'2003* proceedings, UNION Agency, Science Press, Plzeň, 2003.

[Mic03] **Microsoft COM Technologies**.
http://www.microsoft.com/com/

[MSDN02] **MSDN (Microsoft Developer Network) Library**.
http://msdn.microsoft.com/library/

[Her03] **Centre of Computer Graphics and Data Visualisation**.
http://herakles.zcu.cz/research.php

[Vis03] **Visual Studio .NET Documentation**.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vsstartpage.asp