I., Skala,V.: OpenGL interface for .NET, EC&I 2004, Košice, VIENALA Press, pp.421-426, ISBN 80-8073-150-0, 2004

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS
OF
THE TECHNICAL UNIVERSITY OF KOŠICE

PROCEEDINGS
of
THE SIXTH INTERNATIONAL SCIENTIFIC CONFERENCE

ELECTRONIC COMPUTERS and INFORMATICS ECI 2004

The conference is organized by
Department of Computers and Informatics

In co-operation with:
Slovak Society for Applied Cybernetics and Informatics
Czech and Slovak Society for Simulation

Sponsoring by:
SIEMENS Program & System Engineering s.r.o. Bratislava
Ing. Milan Roško, TEGH, Information Technology Dep. Toronto, Kanada
ZTS Výskumno-vývojový ústav a.s., Košice

Editors: Štefan Hudák, Ján Kollár

September 22-24, 2004
Košice - Herľany
Slovakia

V.: OpenGL interface for .NET, EC&I 2004, Košice, VIENALA Press, pp.421-426, ISBN 80-8073-150-0, 2004

Editors' Note

This publication was reproduced by the photo process, using the manuscripts and soft copies supplied by their authors. The layout, figures, and tables of some papers did not comform exactly with the standard requirements. In some cases the layout of the manuscripts were rebuilt. All mistakes in manuscripts there either could not be fixed, or could not be checked completely by reviewers and there are a responsibility of authors. The readers are therefore asked to excuse any deficiencies in this publication which may have arisen from the above causes.

Proceedings of the Sixth International Scientific Conference Electronic Computers and Informatics ECI 2004
ISBN 80-8073-150-0
Editors: Štefan Hudák, Ján Kollár
September 22-24, 2004, Košice - Herľany, Slovakia

© Layout & Design: J. Bača, D. Mihalyi, D. Sobotová

Additional copies can be obtained from:
Department of Computers and Informatics of FEI,
The University of Technology Košice,
Letná 9, 04200 Košice, Slovakia
Phone: +421-55-63 353 13
Fax: +421-55-6330115
E-mail: Stefan.Hudak@tuke.sk, Jan.Kollar@tuke.sk

Proceedings of
the Sixth International Scientific Conference
Electronics Computer and Informatics ECI 2004

VIENALA Press
Moldavská 8/A, 040 01 Košice
Edition: 55
September 2004

ISBN 80-8073-150-0

Hanák,I., Skala,V.: OpenGL interface for .NET, EC&I 2004, Košice, VIENALA Press, pp.421-426, ISBN 80-8073-150-0, 2004

OpenGL Interface for .NET

Ivo Hanák, Václav Skala
Department of Computer Science and Engineering,
University of West Bohemia, Univerzitní 8, 306 14 Plzeň, Czech Republic
E-mail: hanak@kiv.zcu.cz, skala@kiv.zcu.cz

Abstract

OpenGL is one of the most used interfaces for graphical output that is widely supported by graphical hardware. .NET is an implementation of a platform specification that offers comfortable and safe environment for users. This project provides an interface for .NET and OpenGL cooperation. The interface is aimed on both simplicity and comfort. It provides environment that allows use of strictly managed code for generating output with hardware support. This paper describes briefly major features of OpenGL, .NET, and introduces our solution. Description of both benefits and drawbacks of the solution is provided. At the end, a performance comparison with other existing solutions is presented.

1. Introduction

.NET environment is a set of binary utilities that allows the developer to execute a code in a safe and comfortable environment. It allows the user to create platform independent code even on binary level. A part of the environment is a set of routines in a form of a library. This library provides quite large scale of functionality. However, it supports only basic 2D output based on GUI capabilities.

This paper presents brief description of our approach for OpenGL and .NET environment interoperability. Rather then complete solution this paper present feasibility study of the approach that is aimed on a comfort from a viewpoint of the user.

In the Section 1, this paper describes environments and major tools that were used for creating of the solution. Section 2 then contains brief introduction to another already available solution. Difficulties that are solved by the project are described in Section 3. Implementation is described in Section 4 and results in a form of performance comparisons are presented in Section 5.

1.1 .NET Environment

.NET environment is an object oriented runtime environment whose retail version was produced in year 2002 by Microsoft. It is an implementation of ISO/ECMA standard that specifies an environment for a platform independent code, i.e. Common Language Infrastructure (CLI) [1] that consists of two major parts: Common Type System (CTS) and Class Library, which specifies an amount of routines supported by any correct *CLI* implementation.

The *CTS* specify support for various data types including operations that are available to programming languages [2]. Every language that is able to fit in the *CTS* capabilities may be used to generate code that is compatible with any CLI implementation. The CLI is not bounded to any specific programming language and there are available languages, such as C#, J#, Visual Basic.NET, C++ Managed Extension (MC++), Ada.NET, etc. The construction of the CLI allows easy cooperation of a code generated by any of CLI-compatible languages.

The CLI provides a safe and comfortable environment. It uses memory management in a form of a *garbage collector* (GC) [8]. It also supports security that is based on permissions bounded to an assembly. *Assembly* is the smallest distributable code possible. Permissions are based on the origin of the code and allow the user to restrict possible dangerous operations.

The code that is run by CLI implementation is called a *managed code*. Such code is stored in a form similar to machine code, i.e. Common Immediate Language (CIL). The CIL is rather optimized

Hanák,J., Skala,V.: OpenGL interface for .NET, EC&I 2004, Košice, VIENALA Press, pp.421-426, ISBN 80-8073-150-0, 2004

for a compilation and the CLI utilizes Just-In-Time compiler for this purpose. The CLI supports execution of native binary code, i.e. *unmanaged code*. The unmanaged code cooperation is supported in a form of:

- *Platform Invoke* (P/Invoke) that allows executing simple functions stored in a DLL library.
- *COM Interoperability* that allows a user to operate with COM object.
- *It Just Works* (IJW) mechanism that allows fine control over data type conversions and therefore leads to better performance in the comparison to P/Invoke. However, this mechanism is available only from MC++.

1.2 OpenGL

OpenGL is common graphic interface developed by SGI [3]. It allows advanced graphical output and utilizes hardware if available. The interface consists of simple functions and numeric constants. The fact that the interface consists only of functions stored in a DLL allows us to benefit from CLI capabilities for unmanaged code cooperation. However, the interface utilizes pointers quite a lot and therefore the direct introduction of these functions to the managed environment is not suitable.

Functionality of the OpenGL library can be extended and/or simplified by other libraries such as the GLU [4] and the GLUT. The GLU contains set of functions that allows rendering of high-order primitives, such as parametric surfaces. It also provides support for more sophisticated transformation operation. The GLUT library [5] is a third-party toolkit that simplifies operating system cooperation tasks for the OpenGL.

The essential part of the OpenGL is *GL Extensions* mechanism [6] that allows adding of a new functionality. The system allows anyone to add new functionality by defining new GL Extension. However, each extension may not only to add new function but may also modify functionality and values accepted by already existing function.

Another possibility for high-performance graphical output is the Direct3D (D3D) library. Despite the fact that the D3D provides simple interface that is comfortable at the same time, we aimed on OpenGL from reasons. First, the D3D has already offi managed version. Second, the D3D currently available only on a single platfor

1.3 The Goal

The goal the project is to cr interface that would allow advanced graph output with hardware support. This interf shall use native binary libraries, such OpenGL, in order to provide desi functionality. It is clear that for such purp the best solution is to port some other alre available interface because it leads to lower requirement on learning time for s interface.

The next goal of the project is provide simple and easy interface, interface that would be possible to use with deep knowledge of .NET environment. Tha to that the user should be able to experim with the interface without long and extens learning. The interface shall not force use use pointers and low level native c operations because the use of mentio capabilities requires knowledge that is dee than the necessary level.

The interface shall also utilize enu instead of numeric constants. It narrows do the set of values the user is forced to cho from. It shall also prevent from accider passing of wrong value as a funct parameter.

One of the important goals of project is to allow parameter checking in or to prevent unexpected application crashes, e accessing invalid memory. It shall also prev the user from passing invalid combination parameter values.

The last goal the project is performance. We are sure that there will some performance loss due to both .N framework environment and unmanaged co cooperation. However, the loss shall be as l as possible and the result shall be compara to other solution and even to unmanag version.

Hanák,I., Skala,V.: OpenGL interface for .NET, EC&I 2004, Košice, VIENALA Press, pp.421-426, ISBN 80-8073-150-0, 2004

2. Previous Work

One of the already existing ports of the OpenGL to the .NET is the CsGL project [7]. This project is a major and the most finished one. It provides an exact and complete copy of the OpenGL 1.4 and the GLU 1.4 interface considering managed code restriction. All functions and constants are static members of a single class. All functions and constants of every GL Extensions are members of the same class. Due to the inheritance, the final class contains all OpenGL functions that are available.

The recommended use of the CsGL is via inheritance. Thanks to that the user can access all OpenGL functions in the manner similar to plain C. However, the user can separate functions that belong to different GL Extensions only by their identifiers. Important advantage of the CsGL is that it allows automatic creation of ports of newer OpenGL version. The implementation is done in C and C#. It utilizes P/Invoke mechanism and tries to avoid pointers. The CsGL utilizes its own special classes to solve the problem. The drawback is the fact that the user is forced to use CsGL types instead of CLI native types in order to avoid unsafe code.

3. Difficulties

All difficulties that we have to solve are based on CLI capabilities, i.e. properties of managed environment. Major difficulties are: data sharing, callback functions, and void pointers.

Data sharing is one of the basic functionality provided by the interface. Value data type sharing is no difficulty at all because it does not require almost any conversion. On the other hand, reference data types, such as arrays, are the difficulty. Allocation and destroying of the memory for these data types is managed by the GC. This facility is able to destroy allocated block of memory if there is not reference to such block.

From the viewpoint of the managed system, pointers used by unmanaged code are just integers. Whenever the user deletes all reference to particular memory block, the GC can destroy such block even though the unmanaged part still uses (unmanaged) pointer to such block.

The next capability of the GC is to prevent too large memory fragmentation. In order to avoid the situation, the GC is able to move block of a memory. The GC moves blocks with no concern about the possible unmanaged pointer, i.e. the unmanaged pointer is not updated.

Callback functions are the next important difficulty. They are used by the GLU in order to provide error notification and/or solving of special situation during high-order primitive tessellation. The problem is based on the fact that these callbacks requires pointer to a function. The user shall be able to use even non-static member function of the class for such callback.

The last difficulty is *void pointer*. This difficulty is based on a fact that the solution shall prevent forcing the user to use unsafe blocks and/or replacement for native built-in data types. It is not possible to provide straightforward replacement of the void pointers by any of valid managed construction.

4. Solution

Our solution is a wrapper that utilizes native binary libraries of the OpenGL and benefits from .NET capabilities of unmanaged code cooperation. This allows the user to use functionality stored in the unmanaged native binaries of the OpenGL without need of an additional structures provided by the user. The implementation uses MC++ in order to benefit from the IJW.

4.1 Structure

Based on features of the .NET environment we utilized object oriented approach. All functions are divided into groups based on their affiliation to particular OpenGL versions, GLU versions, and GL Extension. We avoid further division because that would modify the interface too much in comparison to the original OpenGL. We then create these groups of classes:

- *System classes* that handles initialization and underlying GUI cooperation.

Hanák I., Skala,V.: OpenGL interface for .NET, EC&I 2004, Košice, VIENALA Press, pp.421-426, ISBN 80-8073-150-0, 2004

- *OpenGL and GLU classes* contain functions of both OpenGL and GLU interface. These classes require system classes. New versions of the interface are derived from these classes.
- *GL Extension classes* provide functionality of GL Extensions. Functions are divided by their affiliation to particular class. Functions modified by particular GL Extension are included too. The advantage of this approach is that the user is able to choose which extensions to use. On the other hand this approach is not useful for complicated extensions that modify parameters of far too many functions.
- *Internal data structures* that contain data kept inside of OpenGL/GLU for later use, e.g. vertex arrays. It provides protection from the GC. The drawback is the fact that these classes may require modification when new version of the OpenGL is wrapped. However, these classes are invisible to the user and therefore the interface stays intact.
- *Other classes* that provide functionality similar to GLUT, i.e. they simplify underlying GUI cooperation.

Constants that are used by the interface are replaced by enums. The division into particular enums is based on their affiliation to functions. The advantage of enums is a fact that the user is not forced to choose from wide range of constants only a small set is available. It allows the user to benefit from capabilities of modern IDEs that provide completion of identifiers. The disadvantage is a fact that this leads to the code that differs from original OpenGL code. Example of a simple code is available in Figure 1.

```
gl.ClearColor(0, 0, 0, 0);
gl.Clear(
     ClearMasks.GL_COLOR_BUFFER_BIT);
gl.Begin(RenderPrimitive.GL_POINTS);
gl.Color3d(1, 1, 1, 1);
gl.Vertex3d(0, 0, 0);
gl.End();
```

Figure 1: Example of a code that draws a point

4.2 Construction

Among other things, the wrapper allows us to implement *parameter checking* that prevents the user to use invalid and/or uninitialized structures. It can also protect the user fro passing invalid combination of parameters. is sure that this may lead to significa slowdown. However, it is possible to crea two versions of the interface: one that does n utilize parameter checking and one that doe The last version is intended for debuggi purposes.

As it was mentioned each functi wraps another existing function. This wrapp first performs parameter checking and th locks data to protect them if required. As ne the wrapper calls original function and obtai its return value that is stored into tempora variable. Afterwards the wrapper unloc locked data and returns the return value if an

4.3 Difficulties

One of the major difficulties is the d sharing of reference types, i.e. protection referenced data structures from the GC. order to solve the difficulty we have th possible approaches:

- *P/Invoke* is a general approach intend for calling unmanaged functions stored DLL. During the call, the P/Invoke hand protection from the GC and the d conversion if required. The P/Invoke is second fastest approach in comparison the rest. It is also capable to han callbacks. Therefore our solution uses for such purpose.
- *GCHandle* is a structure that allo handling of almost any reference data ty However, the *GCHandle* is slow due runtime validity checking. On the otl hand, it is a general approach that allo us to handle even data that are sto inside OpenGL binary for later use, vertex arrays. Therefore our soluti utilizes *GCHandle* for such purpose.
- *Pinning pointers* that are similar *GCHandle* but with a few restrictions. major restriction is the fact that the pinn pointer data type can be used only a local variable. On the other hand, pinn pointers are significantly faster then *GCHandle* because they do not util runtime validity checking. Pinning point are used for data that are not stored ins OpenGL for later use.

- *Structures.* In a few cases the reference data type can be replaced by a value type because the function utilizes a pointer in order to read data stored inside predefined structures, e.g. *glVertex2dv*. Our solution utilizes structures wherever possible.
- *.NET provided capability.* In a few cases the .NET environment has a capability similar to the *GCHandle*. This approach is even faster then pinning pointers and needs only a little additional constructions. Therefore this approach is the most preferred.

The next difficulty we had to solve is called *void pointer*. In this case we tried to avoid *IntPtr* structure that may be replacement for a general pointer. However, this structure is just an integer form the viewpoint of the environment and it requires additional constructions and/or function calls in order to obtain it. Therefore we used either general arrays or overloading.

General arrays benefits from the fact that each managed array is derived from a single class. This class may be used as a replacement for the void pointer. However, only *GCHandle* can be used for that class and therefore this approach is not preferred.

Overloading is the most preferred approach. It is based on a fact that the void pointer is used for sharing of general data structures where the description is provided by other parameters of the function. Therefore the *void pointer* can be replaced by appropriate data type based on other parameters. In this case our solution benefits from parameter checking in order to prevent passing of invalid combination of parameters.

5. Results

Once we implemented our interface we wanted to know its performance. For this purpose we created set of tests that allow us to check the performance of possible bottlenecks. We compared our solution to the CsGL as another major OpenGL port for managed environment.

The results of tests are relative to unmanaged version, i.e. Win32 version. This means that values smaller then 1.0 denote worse performance. In order to make our tests complex we compared both debug, i.e. our implementation with parameter checking, and retail version, i.e. our implementation without parameter checking.

First test is aimed on passing of built-in data value. The test renders a triangle that is created by calling of a *glVertex2d* function. The test utilizes only functions that use built-in values. The results are available in Figure 2. The performance of unmanaged and both managed versions are very close and there are not almost any performance losses.

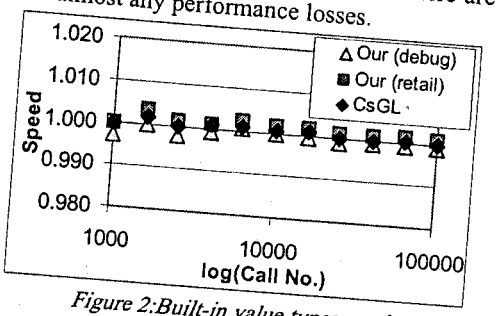

Figure 2:Built-in value types result

Next test compares *GCHandle* and P/Invoke mechanism. It utilizes texture setting via *glTexImage2D* function. The CsGL version uses P/Invoke while our solution utilizes the *GCHandle* in order to consume general arrays. For the results consults the Figure 3.

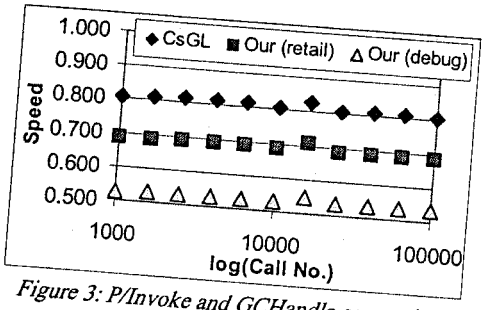

Figure 3: P/Invoke and GCHandle comparison

As it is clearly visible, the *GCHandle* is slower then the P/Invoke. This is caused by the additional runtime checks performed by the *GCHandle* structure. However, such functions are not called too often and therefore the overall impact shall not be significant. For more details refer to the last test.

The *next test* compares approaches for data sharing of smaller block of memory, i.e. structures. For such purpose a *glColor4fv* function is used. This test compares P/Invoke with and without unsafe block and our solution using structures and pinning pointers. In this

case, only retail version of our solution is compared. The results are shown in Figure 4.

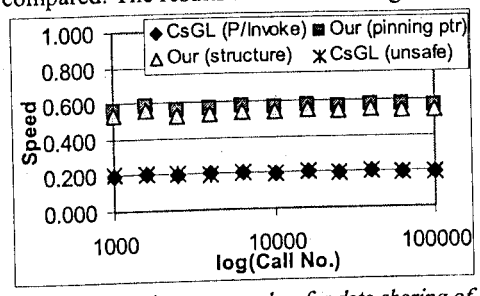

Figure 4: Various approaches for data sharing of non-built-in value data types

From the results it is sure that the existence of unsafe blocks has almost no impact on P/Invoke performance. Also the performance of structures, i.e. value data types, and pinning pointers is very close to each other and therefore solution that utilizes value data types instead of arrays shall not suffer from any significant performance dropdown.

The *last test* is a complex scene. This shall check the overall performance for real scene. In this case a matrix of spheres is rendered. For each sphere its material, texture, transformations, and geometry using vertex array are set. This test combines various approaches to data sharing and the results are available in Figure 5.

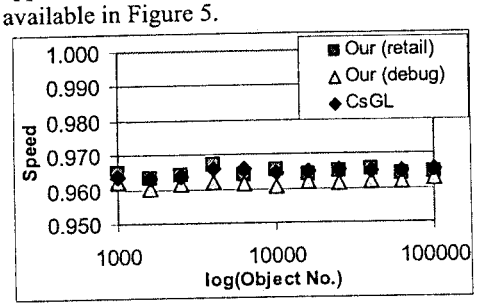

Figure 5: Complex Scene

6. Conclusion

We presented a possible solution for an OpenGL interface in managed environment that is both comfortable and safe. We intended to keep interface as close as it is possible to original OpenGL interface even though we used object oriented approach including inheritance. However, it was not possible to create a smooth object oriented approach due to the construction of the GL Extensions mechanism.

We then proved that our solution provides performance similar to CsGL for complex scene and that both solutions are close to performance of the unmanaged code. It is even possible to assume that there is no significant slowdown for rendering from managed environment. We also implemented a tool that allows semi-automatic generation of wrapper source code from OpenGL C header files. The only manual input is the handling of void pointers. However, it is applied only to a small fraction of the whole function set.

Acknowledgments

We would like to thank you our colleagues and students of APG and GSVD course. This project is supported by the Ministry of Education of The Czech Republic project MSM 23500005 and by Microsoft Research Ltd. (U.K.).

References

[1] Standard ECMA-335: CLI. http://www.ecma-international.org/publications/standards/Ecma-335.htm

[2] Prosise J.: Programming Microsoft .NET. Microsoft Press, 2002.

[3] Segal, M., Akeley, K.: The OpenGL Graphics System: A Specification Version 1.3. http://www.opengl.org/developers/documentation/version1_3/glspec13.pdf

[4] Chin, N., Frazier, C., Ho, P., Liu, Z., Smith, K. P.: The OpenGL Graphics System Utility Library (Version 1.3). ftp://ftp.sgi.com/opengl/doc/opengl1.2/glu1.3.pdf

[5] Kilgard, J. M.: The OpenGL Utility Toolkit Programming Interface (API Version 3). http://www.opengl.org/developers/documentation/glut/spec3/spec3.html

[6] GL Extensions Registry.http://oss.sgi.com/projects/ogl-sample/registry/

[7] CsGL Project. http://csgl.sourceforge.net

[8] Richter, J.: Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework. http://msdn.microsoft.com/msdnmag/issues/1100/GCI/default.aspx.