Implementation of Proxy Libraries with Direct3D for Extraction of Graphics Information

Choong-Gyoo Lim*, Kure-Chel Lee**
*SungKongHoe University, Korea, **ETRI, Korea
E-mail : cglim@skhu.ac.kr*, Gyuchel@etri.re.kr**

1. Introduction

If one can intercept API function calls [1-2] from 3D applications, there are many things that can be accomplished easily by adding more functionalities or replacing with other functionalities. This paper illustrates how to implement API call interception in extracting graphics information from frame images of 3D applications. The graphics information includes the depth value and motion vector of a screen pixel and an object's boundary. For the purpose of the current research, it suffices to implement API call interception for Direct3D API because frame rendering of applications is performed by calling its API functions. Proxy libraries are constructed to implement API call interception. The current implementation is based on the Direct3D 9.0c that was released in April 2006. It is successfully applied to some examples of its SDK.

2. API call interception

2.1. Classification of Direct3D drawing functions

Direct3D consists of its core library and extension library, of which each is built with various kinds of COM objects. Thus, call interception is necessary for member functions of the COM objects because most function calls are of member functions of COM objects. The libraries additionally include some global functions that need call interception as well.

2.2. COM object member functions

Firstly, we define wrapper classes for COM objects and also member functions corresponding to member functions of COM objects. Because the wrapper functions (member functions of wrapper classes) call the member functions of COM objects, the wrapper class is defined to inherit COM objects. For example, the 'Direct3D9Wrap' class is defined with 'Direct3D9' class as a parent, which is the first COM object that Direct3D-based 3D applications create. The 'Direct3DDevice9Wrap' class also defined, inheriting the 'Direct3Ddevice9' class as a parent, which is another important COM object that 3D applications create.

Because the Direct3D API consists of its core library and extension library, the proxy libraries are constructed accordingly. The proxy libraries replace DLL (Dynamic-Link Library) on the host Windows system and their backup libraries are loaded together upon the start of applications. When applications call member functions of COM objects, member functions of wrapper classes are called instead.

2.3. Global functions

One of important global functions is IDirect3D9 *Direct3DCreate9(UINT) which create a Direct3D object and returns its address. Accordingly, a wrapper function is defined to create an object of the wrapper class 'Direct3D9Wrap' and returns its address. The member function HRESULT IDirect3D9::CreateDevice(UINT, D3DDEVTYPE, HWND, DWORD, D3DPRESENT_PARAmeTERS *, IDirect3DDevice9 **) creates an object of the 'Direct3DDevice9' class and returns its address. The corresponding member functions of the wrapper class 'Direct3D' creates an object of the 'Direct3DDevice9Wrap' class and returns its address. For the Direct3D core library, Direct3DCreate9() is only global function to be intercepted for our purpose.

Some of global functions of the extension library are needed to be intercepted. They include D3DXLoadMeshFromX(), D3DXCreateTextureFromFile(), D3DXCreateFontIndirect(). We define wrapper functions to be called instead of the original functions. The wrapper function of D3DXLoadMeshFromX() creates an object of the wrapper class 'Direct3DXMesh9Wrap' that inherits the 'ID3DXMesh' class and returns its address. The addresses of the original functions from the backup libraries are saved so that the wrapper functions can call the original ones.

3. Extraction of graphics information

The graphics pipeline computes depth values of screen pixels directly and saves into its depth buffer (or Z-buffer). However, the Direct3D API does not allow a direct access to the buffer and thus makes extraction of depth values a difficult task until the release of 10.0 [3]. We utilize the off-screen rendering [4-5] where a dynamically-created texture replaces the color buffer of the render target temporarily so that depth values are stored as color values.

The wrapper functions save the calling information to the command queue as illustrated in **Figure** *1*. All calling information of Direct3D API functions for each frame is stored. The wrapper functions call the original API functions as well to construct the frame image. It restores the calling information and calls the original API functions to compute graphics information while simultaneously providing the pipeline with a customized shader program to compute depth values. Similarly, pixel motion vectors and object boundary can be computed using the same off-screen rendering technique with customized shader programs as in computing pixel depth values.

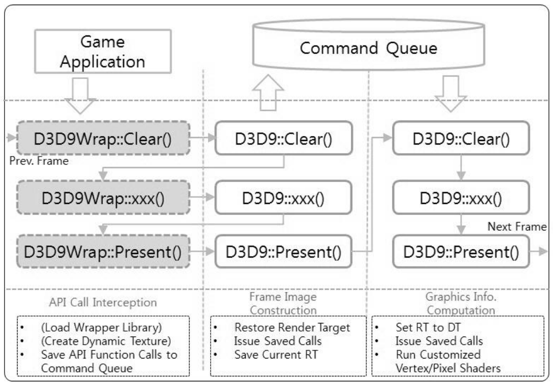

Figure 1. Control flow of a frame

4. Conclusion

Figure *2* shows the results of extracting graphics information from a sample 3D application. The overhead is minimal even with one more round of API function calls for each frame and saving and retrieving the calling information to the command queue. The current implementation can show the extracted graphics information on the screen at the same time without much overhead.

By implementing API call interception, the graphics information is extracted without any modification to the source codes of applications. It thus incurs no development cost, which is hardly feasible otherwise. These graphics information can be utilized to encode consecutive frame images of 3D applications such as computer games because it can provide an encoder with a good information on where similar blocks are located and thus allow the encoder to achieve a better video compression ratio [6].

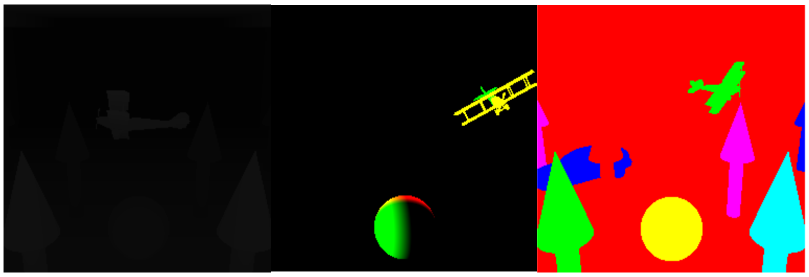

Figure 2. Results of graphics information extraction (L: depth value, M: motion vector, R: object boundary)

5. References

[1] "API Hooking", Accessed Oct. 18, 2012, http://en.wikipedia.org/ wiki/Hooking.
[2] S. Kim, "Intercepting System API Calls", Mar. 7, 2012, Accessed Oct. 18, 2012, http://software.intel.com/en-us/articles/intercepting-system-api-calls.
[3] I. Cantlay, "High-speed, Off-screen Particles", GPU Gems 3, H. Nguyen, Ed., Nvidia, 2007, pp.513-528.
[4] Chris Wynn, "Using P-Buffers for Off-Screen Rendering in OpenGL", NVidia Technical Paper, Nvidia, 2002, Accessed Nov. 25, 2013, https://developer.nvidia.com/sites/default/files/ akamai/ gamedev/docs/PixelBuffers.pdf.
[5] K. Harris, "Off-screen Rendering", Direct3D (DirectX 9.0) Code Samples, Feb. 11, 2005, Accessed Dec. 1, 2013, http://www.codesampler.com/dx9src/dx9src_7.htm.
[6] I. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia", Wiley, 2003.