IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
AUSTIN DIVISION

LUCIO DEVELOPMENT LLC,	§	
	§	
Plaintiff,	§	Case No: 1:17-cv-1154
	§	
vs.	§	PATENT CASE
	§	
NVIDIA CORPORATION,	§	
	§	
Defendant.	§	
	§	

COMPLAINT

Plaintiff Lucio Development LLC ("Plaintiff" or "Lucio") files this Complaint against NVIDIA Corporation ("Defendant" or "NVIDIA") for infringement of United States Patent No. 7,069,546 (hereinafter "the '546 Patent").

PARTIES AND JURISDICTION

1. This is an action for patent infringement under Title 35 of the United States Code. Plaintiff is seeking injunctive relief as well as damages.

2. Jurisdiction is proper in this Court pursuant to 28 U.S.C. §§ 1331 (Federal Question) and 1338(a) (Patents) because this is a civil action for patent infringement arising under the United States patent statutes.

3. Plaintiff is a Texas limited liability company with its office address at 555 Republic Dr., Suite 200, Plano, Texas 75074.

4. On information and belief, Defendant is a Delaware corporation having a place of business at 2701 San Tomas Expressway, Santa Clara, CA 95050, with a Regional Office at 11001 Lakeline Blvd #100, Austin, TX 78717. On information and belief, NVIDIA is

registered to do business in the State of Texas and may be served with process by delivering a summons and a true and correct copy of this Complaint to its registered agent for receipt of service of process, Corporation Service Company, 211 E. 7th Street, Suite 620 Austin, TX 78701.

5. This Court has personal jurisdiction over Defendant because Defendant has committed, and continues to commit, acts of infringement in this District, has conducted business in this District, and/or has engaged in continuous and systematic activities in this District.

6. On information and belief, Defendant's instrumentalities that are alleged herein to infringe were and continue to be used, imported, offered for sale, and/or sold in this District.

VENUE

7. Venue is proper in this District pursuant to 28 U.S.C. §1400(b) because acts of infringement are occurring in this District and Defendant has a regular and established place of business in this District. For instance, on information and belief, Defendant has a regular and established place of business at 11001 Lakeline Blvd #100, Austin, TX 78717.

COUNT I
(INFRINGEMENT OF UNITED STATES PATENT NO. 7,069,546)

8. Plaintiff incorporates paragraphs 1 through 7 herein by reference.

9. This cause of action arises under the patent laws of the United States and, in particular, under 35 U.S.C. §§ 271, *et seq*.

10. Plaintiff is the owner by assignment of the '546 Patent with sole rights to enforce the '546 Patent and sue infringers.

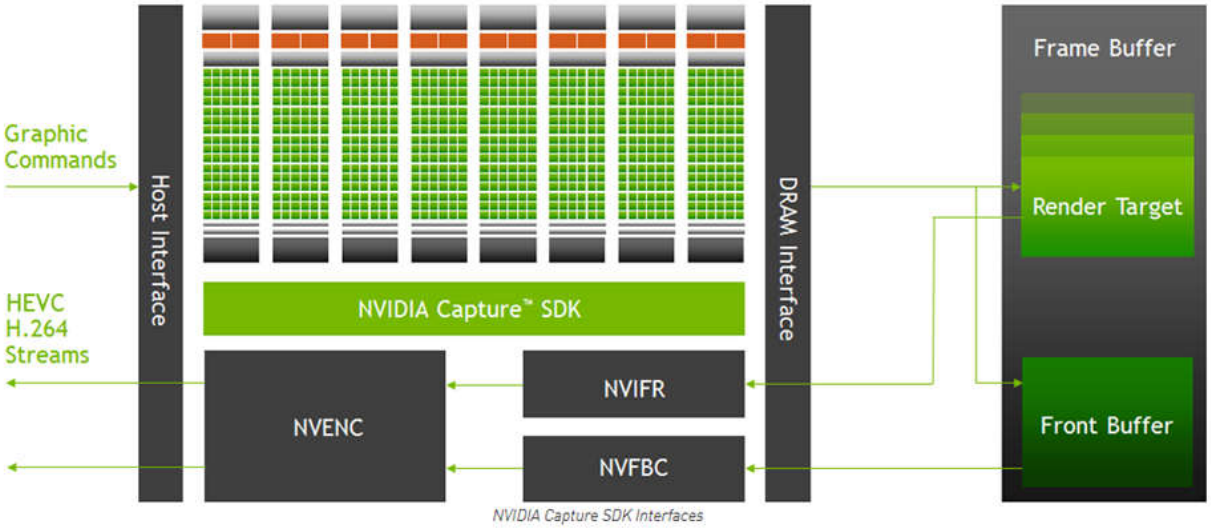
11. A copy of the '546 Patent, titled "Generic Framework for Embedded Software Development," is attached hereto as Exhibit A.

12. The ‘546 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code.

13. On information and belief, Defendant has infringed and continues to infringe one or more claims, including at least Claim 1, of the ‘546 Patent by making, using, importing, selling, and/or offering for sale a software platform for embedded software development, which is covered by at least Claim 1 of the ‘546 Patent. Defendant has infringed and continues to infringe the ‘546 Patent directly in violation of 35 U.S.C. § 271.

14. Defendant, sells, offers to sell, and/or uses embedded software development packages including, without limitation, the NVIDIA Capture SDK, and any similar products (“Product”), which infringe at least Claim 1 of the ‘546 Patent.

15. The Product is a framework (e.g., a software development kit) that is configured to create embedded software for multiple hardware modules. For example, the Product is a programmable software development kit (SDK) for multiple operating systems (Such as Linux, Windows and Mac) and GPU (Graphic Processing Unit) hardware modules such as NVIDIA Quadro 2000 class or higher, NVIDIA Tesla and/or other kits supported by NVIDIA. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.



NVIDIA Capture SDK Interfaces

Source: https://developer.nvidia.com/capture-sdk

The NVIDIA Capture Software Development Kit, previously called as GRID SDK is a comprehensive suite of tools for NVIDIA GPUs that enable high performance graphics capture and encoding. This Programming Guide describes how to use the various NVIDIA Capture SDK interfaces available on GRID, Quadro, and specific Tesla Products.

1.1 GPU ACCELERATED READBACK AND ENCODE

The NVIDIA Capture SDK includes two API interfaces for high performance readback of rendered content from the GPU and video encoding on the GPU:

1.1.1 NVFBC - NVIDIA Framebuffer Capture

The NVIDIA Framebuffer Capture (NVFBC) API captures and optionally compresses the entire Windows desktop or full-screen applications running on the supported Operating Systems (For list of Operating Systems, please refer to the SDK release notes). It essentially provides the same output as a real connected monitor to the GPU: a full desktop, with application windows, menu bar, composited overlay and hardware cursor. As such, NVFBC is ideally suited to *desktop capture and remoting*.

NVFBC has many advantages over existing methods of framebuffer capture. It is resilient to Aero DWM (enable/disable) changes and resolution changes. It operates asynchronously to graphics rendering because it is able to use the dedicated hardware compression and copy engines on the GPU. It delivers frame data to system memory faster than any other display output or other readback mechanisms all while having minimal impact on the rendering performance.

Source: https://developer.nvidia.com/capture-sdk

1.1.2 NVIFR - NVIDIA Inband Frame Readback

The NVIDIA Inband Frame Readback (NVIFR) API captures and optionally compresses an individual DirectX or OpenGL graphics render target. Unlike NVFBC, the output from NVIFR does not include any window manager decoration, composited overlay, cursor or taskbar; it solely provides the pixels rendered into the render target, as soon as their rendering is complete, ahead of any compositing that may be done by the windows manager. In fact, NVIFR does not require that the render target even be visible on the Windows desktop. It is ideally suited for *application capture and remoting*, where the output of a single application, rather than the entire desktop environment, is captured.

NVIFR is intended to operate *inband* with a rendering application, either as part of the application itself, or as part of a shim layer operating immediately below the application. Like NVFBC, NVIFR operates asynchronously to graphics rendering, using dedicated hardware compression and copy engines in the GPU, and delivering pixel data to

1.1. Overview

1.1.1. CUDA Programming Model

The CUDA Toolkit targets a class of applications whose control part runs as a process on a general purpose computing device, and which use one or more NVIDIA GPUs as coprocessors for accelerating *single program, multiple data* (SPMD) parallel jobs. Such jobs are self-contained, in the sense that they can be executed and completed by a batch of GPU threads entirely without intervention by the host process, thereby gaining optimal benefit from the parallel graphics hardware.

The GPU code is implemented as a collection of functions in a language that is essentially C++, but with some annotations for distinguishing them from the host code, plus annotations for distinguishing different types of data memory that exists on the GPU. Such functions may have parameters, and they can be called using a syntax that is very similar to regular C function calling, but slightly extended for being able to specify the matrix of GPU threads that must execute the called function. During its life time, the host process may dispatch many parallel GPU tasks.

Source: http://docs.nvidia.com/cuda/pdf/CUDA_Compiler_Driver_NVCC.pdf

Key Features of NVIDIA Capture SDK

Capture the full desktop (NVIDIA Frame Buffer Capture aka NvFBC)

NvFBC works by capturing the entire contents of the desktop to a GPU buffer without stalling, or interfering with, the other work on the GPU. On Windows, the capture can occur with the Windows Desktop Manager enabled or disabled. This buffer can then be encoded using H.264 or HEVC via on-chip hardware video encoder through the NvEncode API.

Source: http://www.nvidia.com/object/sdk-9.html

What's new in NVIDIA Capture SDK 6.1

- Support for Pascal based Tesla GPUs
- Support for configurable NVFBC diffmap block size (Windows)
- Memory Optimization, footprint reduced by around 15% (Windows)
- RGB888 output support (Linux)
- Addition of time-out for blocking NvFBC capture (Linux)

Operating System	Windows 7, 8, 10, Windows Server 2008/2010/2012 Linux
Dependencies	For Kepler/Maxwell/Pascal generations: all NVIDIA Quadro 2000 class or higher are supported and all NVIDIA Tesla are supported *note: Fermi based GPUs are not supported,* *K1/K2/K520/K340 are not supported* **Linux drivers 384.59 or newer** **Windows drivers 385.05 or newer**
Development Environment	Windows: Visual Studio 2008/2010/2013 Linux: gcc 4.8 or higher
Graphics APIs	DirectX9,10,11 and OpenGL

Source: https://developer.nvidia.com/capture-sdk

16. The Product provides one or more generic application handler programs. For example, NVIDIA Capture SDK provides one or more generic application handler programs such as NVFBC (NVIDIA Framebuffer Capture) and NVIFR (NVIDIA Inband Frame Readback) includes code and libraries which are common and uniform across a plurality of supported NVIDIA GPUs. The generic programs comprise computer program code for

performing generic application functions common to multiple types of hardware modules used in a communication environment (e.g., the generic code provides common and generic functions to multiple hardware modules, as previously identified in paragraph 15). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

NVIDIA Framebuffer Capture (NVFBC) is a high performance, low latency API for reading back display frames from one or more GPU *display heads*. NVIDIA GPUs typically support at least two display heads, and these are usually associated with a physical display output such as a DVI, DisplayPort, or HDMI connector. NVFBC provides essentially the same output one would see on a monitor connected to the GPU: a full desktop, with application windows, menu bar, composited overlay and hardware cursor. By operating asynchronously to graphics rendering and using dedicated hardware compression and copy engines in the GPU, NVFBC delivers frame data to CPU-based applications faster than any other display output or readback mechanism, with minimal impact on rendering performance.

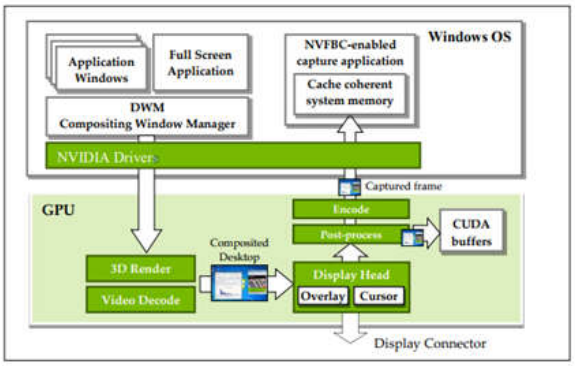


Figure 1 NVFBC framebuffer capture

2.1 HEADER FILES AND CODE SAMPLES

This manual provides an overview of how to use NVFBC. Further details are contained in the NVFBC header files and code samples that are included in the NVIDIA Capture SDK Toolkit:

NVFBC header files are installed in `%CAPTURESDK_PATH%\inc\NVFBC`. All NVFBC applications should include one or more of the mode-specific NVFBC header files, depending on the functionality desired:

Header file	Description
`NVFBC.h`	Top level header file included by all NVFBC applications
`NVFBCToSys.h`	Defines ToSys interface; reads back uncompressed frames to system memory.
`NVFBCCuda.h`	Defines Cuda interface; reads back uncompressed frames to CUDA-mapped buffers in the GPU's framebuffer.
`NVFBCToDx9vid.h`	Defines the DX9Vid interface; reads back uncompressed frames to D3D9-mapped buffers in the GPU's framebuffer.
`NVFBCHWEnc.h`	Defines the capture+encode interface; reads back compressed video frames to system memory. Compression is performed using NVENC HW Encoder engine. Supports H.264 and HEVC compression.
`NVHWEnc.h`	Definitions for NVENC HW Encoder configuration settings, to be included with NVFBCHWEnc.h in the application.

Table 1 NVFBC header files

The following NVFBC code samples are installed in `%CAPTURESDK_PATH%\samples\`

Please refer to the NVIDIA Capture SDK Samples Description document for details about NVFBC samples included with the SDK.

Source: https://developer.nvidia.com/sites/default/files/akamai/designworks/docs/NVIDIA_Capture_SDK_6/NVIDIA%20Capture%20SDK%20Programming%20Guide.pdf

1.1.2 NVIFR – NVIDIA Inband Frame Readback

The NVIDIA Inband Frame Readback (NVIFR) API captures and optionally compresses an individual DirectX or OpenGL graphics render target. Unlike NVFBC, the output from NVIFR does not include any window manager decoration, composited overlay, cursor or taskbar; it solely provides the pixels rendered into the render target, as soon as their rendering is complete, ahead of any compositing that may be done by the windows manager. In fact, NVIFR does not require that the render target even be visible on the Windows desktop. It is ideally suited for *application capture and remoting*, where the output of a single application, rather than the entire desktop environment, is captured.

NVIFR is intended to operate *inband* with a rendering application, either as part of the application itself, or as part of a shim layer operating immediately below the application. Like NVFBC, NVIFR operates asynchronously to graphics rendering, using dedicated hardware compression and copy engines in the GPU, and delivering pixel data to system memory with minimal impact on rendering performance.

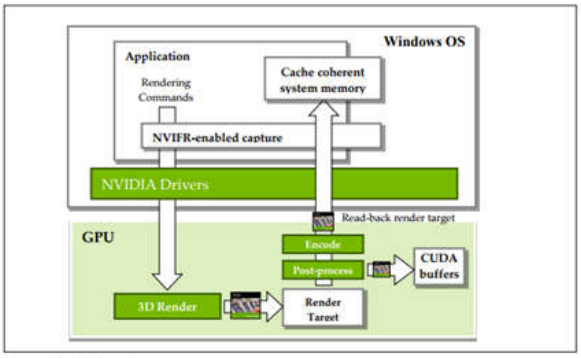


Figure 5 NVIFR render context capture

3.1 HEADER FILES AND CODE SAMPLES

This manual provides an overview of how to use NVIFR. Further details are contained in the NVIFR header files and code samples that are included in the NVIDIA Capture SDK Toolkit:

The NVIFR header files, including interface-specific is installed in `%CAPTURESDK_PATH%\inc\NVIFR\`. All NVIFR applications don't need to include `NVIFR.h` directly, as the specific versions of the NVIFR interfaces include it.

NVIFR code samples are installed in `%CAPTURESDK_PATH%\samples\`

Please refer to the NVIDIA Capture SDK Samples Description document for details regarding the NVIFR samples packaged in the NVIDIA Capture SDK.

Source: http://docs.nvidia.com/cuda/pdf/CUDA_Installation_Guide_Windows.pdf

17. The Product includes generating specific application handler code to associate the generic functions with the specific functions at a device driver for at least one of the types of hardware modules. For example, in addition to the generic application handler code, NVIDIA Capture SDK also includes specific application handler code that is specific to the application (such as applications on Windows, Linux and iOS etc.) and specific to particular GPU hardware (such as GPU display head, display drivers and/or graphic cards supported by NVIDIA). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

Operation of NVFBC is straightforward: after doing one-time setup of the NVFBC API on application load, an application creates an *NVFBC object* for each GPU display head it wishes to read back from, and then enters a processing loop on each NVFBC object to read back frames from each head. Figure 2 provides an overview of the processing flow, which is described in more detail in the following sections.

Note:

[1]This mode works with baremetal, direct attached GPUs, and all vGPU profiles. This is the recommended path when using vGPU profiles that support two or more virtual machines sharing a single GPU. For such vGPU profiles, the CUDA driver is not available. We recommend using this NvFBC path so that capture and encode can be fully accelerated.

[2]This mode is supported in baremetal, direct attached GPUs, and vGPU profiles that limit one virtual machine. The CUDA driver is available and supported in this configuration.

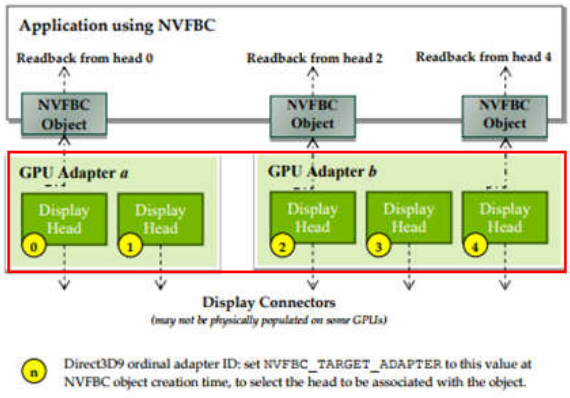


Figure 3 NVFBC objects association with GPU display heads

Source:https://developer.nvidia.com/sites/default/files/akamai/designworks/docs/NVIDIA_Capture_SDK_6/NVIDIA%20Capture%20SDK%20Programming%20Guide.pdf

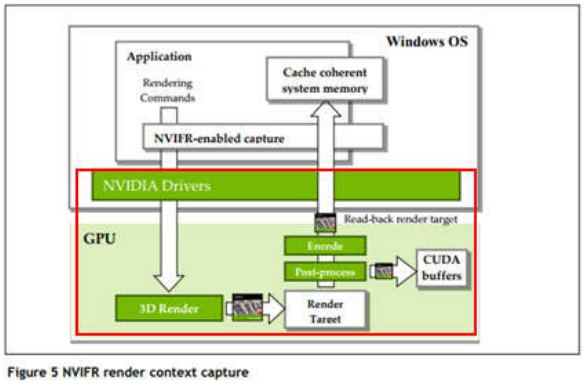


Figure 5 NVIFR render context capture

Source:https://developer.nvidia.com/sites/default/files/akamai/designworks/docs/NVIDIA_Capture_SDK_6/NVIDIA%20Capture%20SDK%20Programming%20Guide.pdf

18. The Product generates specific application handler code and defines a specific element in the specific code to be handled by one of the generic application functions for that hardware module. For example, NVIDIA Capture SDK generates system-specific application handler code by defining a specific element such as functions and data structures corresponding to specific hardware modules (such as NVFBC and NVIFR "objects" for individual GPU display heads) that extend or otherwise connect the system-specific application handler code and data structures made available by the generic application handler code. When specific functions are written for handling defined specific elements, the specific functions must be registered. NVIDIA Capture SDK accordingly contains data structures that register and embed the required functions. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

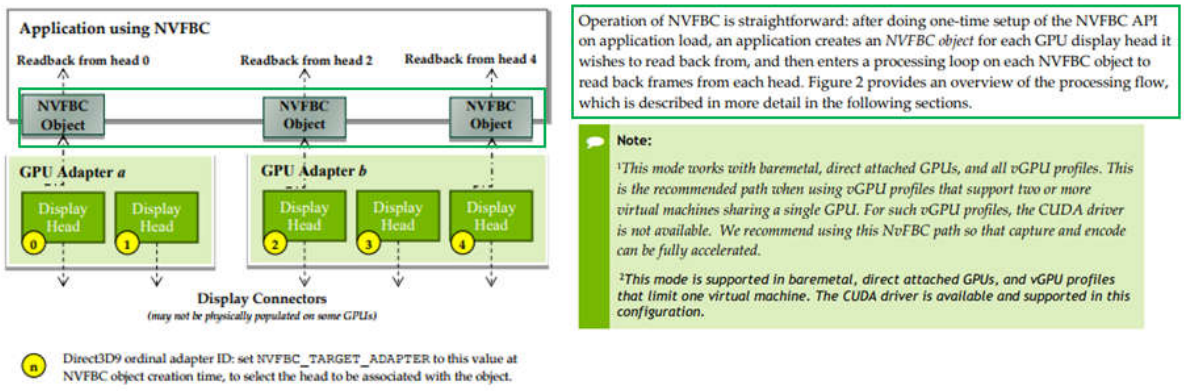


Figure 3 NVFBC objects association with GPU display heads

Source:https://developer.nvidia.com/sites/default/files/akamai/designworks/docs/NVIDIA_Capture_SDK_6/NVIDIA%20Capture%20SDK%20Programming%20Guide.pdf

3.3 CREATING NVIFR OBJECTS

3.3.1 Creating Objects

All NVIFR readback operations are exposed as methods in NVIFR classes. Distinct classes are used to support the different readback modes supported by NVIFR (readback to system memory and readback as compressed video).

To create an NVIFR object you must create an instance of `IDirect3D9Device` or `ID3D10Device` or `ID3D11Device`. This device interface is passed into `NVIFR_CreateEx()` along with the readback format:

```
NVIFRRESULT (__stdcall *NVIFR_CreateFunctionExType) (void *pParams);

// Example usage
ID3D10Device * pDevice = NULL;
D3D10CreateDeviceAndSwapChain(…, &pDevice); // Create the device
NVIFR_CREATE_PARAMS params = {0};
params.version = NVIFR_CREATE_PARAMS_VER;
params.dwInterfaceType = NVIFR_TOSYS;
params.pDevice = pDevice;

NVIFRToSys * toSys = NULL;
NVIFRRESULT res = pfnNVIFR_Create(&params);
if (res == NVIFR_SUCCESS)
{
    toSys = (NVIFRToSys *)params.pNVIFR;
}
```

If successful, the `NVIFR_Create()` call returns a pointer to the newly-created NVIFR object, otherwise it returns `NULL`.

3.4 CAPTURING TO SYSTEM MEMORY

To capture render targets to system memory, create an NVIFRToSys object by specifying `NVIFR_TOSYS` in the `NVIFR_Create()` call:

```
// Create an instance of NVIFRToSys
NVIFRToSys *toSys = pfnNVIFR_Create(device, NVIFR_TOSYS, &version);
```

3.4.1 Setting up the target buffers

`SetupTargetBufferToSys()` must be called before reading back render target buffers.

```
NVIFRRESULT NVIFRSetUpTargetBufferToSys(NVIFR_TOSYS_SETUP_PARAMS
*pParams)

// Example usage
#define NUMFRAMESINFLIGHT = 3
unsigned char *buffer;
HANDLE gpuEvent;

NVIFR_TOSYS_SETUP_PARAMS params = {0};
params.dwVersion = NVIFR_TOSYS_SETUP_PARAMS_VER;
params.eFormat = NVIFR_FORMAT_RGB;
params.eSysStereoFormat = NVIFR_SYS_STEREO_NONE;
params.dwNBuffers = 1;
params.ppPageLockedSysmemBuffers = buffer;
params.ppTransferCompletionEvents =  &gpuEvent;

NVIFRRESULT result = toSys->NVIFRSetUpTargetBufferToSys(&params);
```

Source:https://developer.nvidia.com/sites/default/files/akamai/designworks/docs/NVIDIA_Capture_SDK_6/NVIDIA%20Capture%20SDK%20Programming%20Guide.pdf

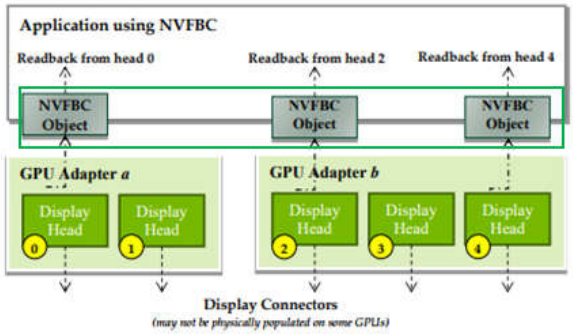


Figure 3 NVFBC objects association with GPU display heads

3.3 CREATING NVIFR OBJECTS

3.3.1 Creating Objects

All NVIFR readback operations are exposed as methods in NVIFR classes. Distinct classes are used to support the different readback modes supported by NVIFR (readback to system memory and readback as compressed video).

To create an NVIFR object you must create an instance of `IDirect3D9Device` or `ID3D10Device` or `ID3D11Device`. This device interface is passed into `NVIFR_CreateEx()` along with the readback format:

```
NVIFRRESULT (__stdcall *NVIFR_CreateFunctionExType) (void *pParams);

// Example usage
ID3D10Device * pDevice = NULL;
D3D10CreateDeviceAndSwapChain(…, &pDevice); // Create the device
NVIFR_CREATE_PARAMS params = {0};
params.version = NVIFR_CREATE_PARAMS_VER;
params.dwInterfaceType = NVIFR_TOSYS;
params.pDevice = pDevice;

NVIFRToSys * toSys = NULL;
NVIFRRESULT res = pfnNVIFR_Create(&params);
if (res == NVIFR_SUCCESS)
{
    toSys = (NVIFRToSys *)params.pNVIFR;
}
```

If successful, the `NVIFR_Create()` call returns a pointer to the newly-created NVIFR object, otherwise it returns `NULL`.

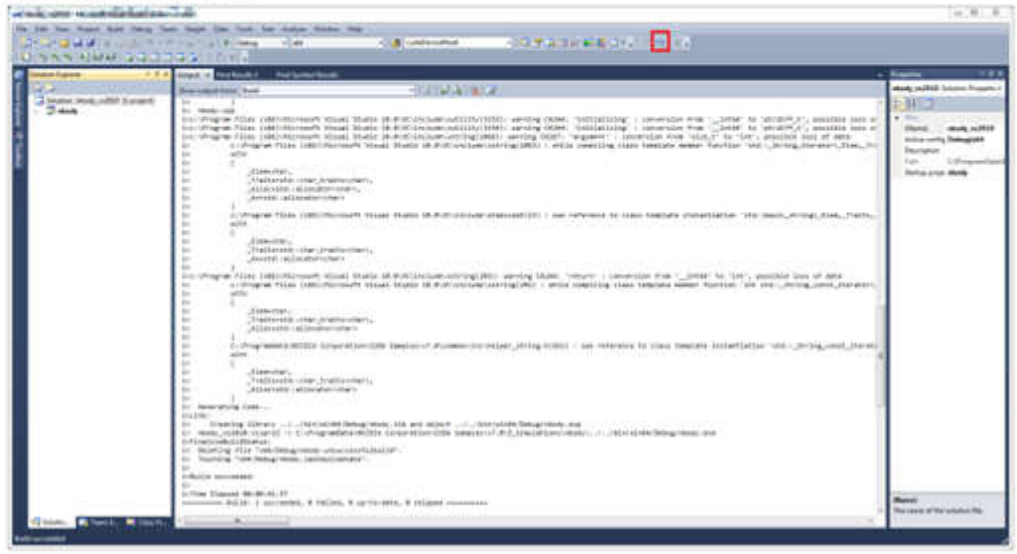
Source:https://developer.nvidia.com/sites/default/files/akamai/designworks/docs/NVIDIA_Capture_SDK_6/NVIDIA%20Capture%20SDK%20Programming%20Guide.pdf

19. When a specific application is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. NVIDIA and/or its customers compile the generic functions and the specific functions using NVIDIA Capture SDK, CUDA compiler and/or any other IDE/Compiler supported by NVIDIA. Certain elements of this limitation are illustrated in the screenshots below and in the

screenshots referenced in connection with other elements herein.

1.1.2. CUDA Sources

Source files for CUDA applications consist of a mixture of conventional C++ host code, plus GPU device functions. The CUDA compilation trajectory separates the device functions from the host code, compiles the device functions using the proprietary NVIDIA compilers and assembler, compiles the host code using a C++ host compiler that is available, and afterwards embeds the compiled GPU functions as fatbinary images in the host object file. In the linking stage, specific CUDA runtime libraries are added for supporting remote SPMD procedure calling and for providing explicit GPU manipulation such as allocation of GPU memory buffers and host-GPU data transfer.



Source: http://docs.nvidia.com/cuda/cuda-quick-start-guide/index.html

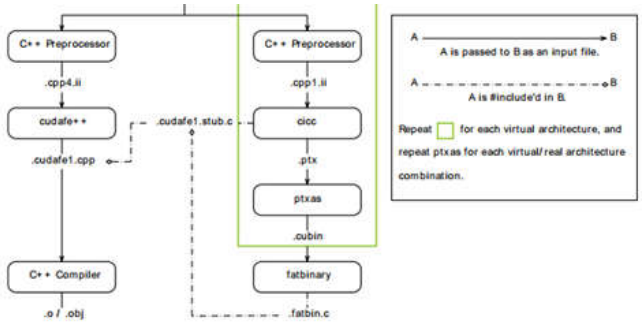
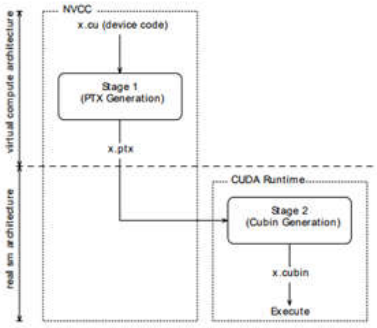


Figure 1 CUDA Whole Program Compilation Trajectory

The compilation step to an actual GPU binds the code to one generation of GPUs. Within that generation, it involves a choice between GPU *coverage* and possible performance. For example, compiling to `sm_30` allows the code to run on all Kepler-generation GPUs, but compiling to `sm_35` would probably yield better code if Kepler GK110 and later are the only targets.



Source: http://docs.nvidia.com/cuda/cuda-quick-start-guide/index.html

20. Defendant's actions complained of herein will continue unless Defendant is enjoined by this court.

21. Defendant's actions complained of herein are causing irreparable harm and monetary damage to Plaintiff and will continue to do so unless and until Defendant is enjoined and restrained by this Court.

22. Plaintiff is in compliance with 35 U.S.C. § 287.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff asks the Court to:

(a) Enter judgment for Plaintiff on this Complaint on all causes of action asserted herein;

(b) Enter an Order enjoining Defendant, its agents, officers, servants, employees, attorneys, and all persons in active concert or participation with Defendant who receive notice of the order from further infringement of United States Patent No. 7,069,546 (or, in the alternative, awarding Plaintiff a running royalty from the time of judgment going forward);

(c) Award Plaintiff damages resulting from Defendant's infringement in accordance with 35 U.S.C. § 284;

(d) Award Plaintiff pre-judgment and post-judgment interest and costs; and

(e) Award Plaintiff such further relief to which the Court finds Plaintiff entitled under law or equity.

Dated: December 11, 2017 Respectfully submitted,

/s/ Jay Johnson
JAY JOHNSON
State Bar No. 24067322
D. BRADLEY KIZZIA
State Bar No. 11547550
KIZZIA JOHNSON, PLLC
1910 Pacific Ave., Suite 13000
Dallas, Texas 75201
(214) 451-0164
Fax: (214) 451-0165
jay@kjpllc.com
bkizzia@kjpllc.com

ATTORNEYS FOR PLAINTIFF

EXHIBIT A