10 AND 12-BIT GRAYSCALE TECHNOLOGY

TB-04631-001_v03 | February 2011

Technical Brief

DOCUMENT CHANGE HISTORY

TB-04631-001_v03

Version	Date	Authors	Description of Change
01	April 17, 2009	SV, SM	Initial Release
02	February 9, 2010	SV, SM	Addition of Table 2
03	February 7, 2011	SV, SM	•Updated “System Specification” section •Updated “Supported Connectors” section •Updated Table 3 and Table 4 •Removed “Moving and Spanning Windows Across Displays” section •Removed “Targeting Specific GPUs for Rendering” section •Added “Directed GPU Rendering” section •Updated “Implementation Details” section

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

10 AND 12-BIT GRAYSCALE TECHNOLOGY

INTRODUCTION

Advances in sensor technology and image acquisition techniques in the field of radiology are producing high bit depth grayscale images in the range of 12 to 16-bit per pixel. At the same time, the adoption of displays with native support for 10 and 12-bit grayscale is growing. These affordable displays are DICOM[1] conformant to preserve image quality and consistency. Furthermore, tiling together multiple such displays enables side-by-side digital study comparisons driven by a single system.

Standard graphics workstations however are limited to 8-bit grayscale, which provides only 256 possible shades of gray for each pixel sometimes obscuring subtle contrasts in high density images. Radiologists often use window-leveling techniques to identify the region of interest that can quickly become a cumbersome and time-consuming user interaction process.

NVIDIA®'s 10–bit and 12-bit grayscale technology allows these high quality displays to be driven by standard NVIDIA Quadro® graphics boards preserving the full grayscale range. By using "pixel packing" the 10-bit or 12-bit grayscale data is transmitted from the Quadro® graphics board to a high grayscale density display using a standard DVI cable. Instead of the standard three 8-bit color components per pixel, the pixel packing allows two 10 or 12-bit pixels to be transmitted, providing higher spatial resolution and grayscale pixel depth as compared to an 8-bit system.

As specialty hardware is not required, NVIDIA's 10-bit grayscale technology is readily available for use with other radiology functions and easy to support amongst a wide range of grayscale panels from various manufacturers. In a preliminary study performed on 10 radiologists using Dome E5 10-bit vs. E5 8-bit displays in conjunction with Three Palms 10-bit, OpenGL accelerated WorkstationOne mammography application, radiologists' performance was statistically significant on the 10-bit enabled display systems, some experiencing triple the read time speedup.

This technical brief describes the NVIDIA grayscale technology, the system requirements and setup. It also aims to guide users through common pitfalls that arise when extending to multi-display and multi graphics processing unit (GPU) environments routinely used in diagnostic imaging and recommends best practices.

Figure 1shows the latest technology in digital diagnostic display systems, a Quadro card driving a 10 mega-pixel, 10-bit grayscale display. Figure 2shows a 10-bit enabled mammography application displaying multiple modalities on multiple displays.

Figure 1. 10 MPixel, 10-Bit Diagnostic Mammography Display[1]

Figure 2. Application Enhanced Using Multiple Displays[2]

[1] Image courtesy of NDS Surgical Imaging, DOME Z10

[2] Image courtesy of Threepalms, Inc.

SYSTEM SPECIFICATION

- 10 and 12-bit grayscale currently requires Windows XP 32-bit and 64-bit
- Windows Vista and Windows 7 supported on R270 or later driver releases
- Grayscale is only supported for OpenGL based applications

Supported Graphics Boards

10-bit grayscale is supported on Quadro graphics boards shown in Table 1. The graphics boards are G80 and higher. The graphics boards are NVIDIA CUDA™ enabled.

Table 1. Quadro Graphics Boards with 10 and 12-Bit Grayscale Support

Mid range – Quadro 2000D, Quadro 2000, Quadro FX 1800 	Recommended for basic 2D image display and manipulation use cases. No auxiliary power is required.
High end – Quadro 4000, Quadro FX 3800, Quadro FX 3700 	Recommended if the primary usage is to display and compute with 2D grayscale images and additionally 3D data,
Ultra high end – Quadro 6000, Quadro 5000, Quadro FX 5800, Quadro FX 4800, Quadro FX 5600, Quadro FX 4600 	Recommended for applications that also require rendering and processing large 3D and 4D geometries and volumes.
Quadro Plex 7000, **Quadro Plex 2200 D2** 	Dedicated deskside visual computing system composed of 2 highest-end Quadro graphics boards with up to 12 GB of total graphics memory. Recommended for advanced visualization and large scale projection and display use cases.

Supported Monitors

The monitor should be capable of 10 and 12-bit outputs. We currently support the following displays.

Table 2. Grayscale Capable Display Panels with Supported Resolution and Pixel Depth

Manufacturer	Panel	Supported Resolutions	Grayscale Depth
NDS Surgical Imaging	Dome E2	•1600 × 1200 at 60 Hz •1200 × 1600 at 60 Hz	10 and 12-bit
	Dome E3	•2048 × 1536 at 60 Hz •1536 × 2048 at 60 Hz	10 and 12 bit
	Dome E5	•2560 × 2048 at 50 Hz •2048 × 2560 at 50 Hz	10 and 12-bit
	Dome Z10	•2560 × 2048 at 50 Hz •2048 × 2560 at 50 Hz •4096 × 2560 at 50 Hz •2560 × 4096 at 50 Hz	10 and 12-bit
Eizo	GS 520	•2560 × 2048 at 50 Hz •2048 × 2560 at 50 Hz	10-bit
NEC	MD205MG, MD205MG-1	•2560 × 2048 at 57 Hz •2048 × 2560 at 57 Hz	10-bit
	MD213MG	•2048 × 1536 at 60 Hz •1536 × 2048 at 60 Hz	10-bit
	MD21GS-3MP	•2048 × 1536 at 60 Hz •1536 × 2048 at 60 Hz	10-bit
	MD21GS-2MP	•1600 × 1200 at 60 Hz •1200 × 1600 at 60 Hz	10-bit
Wide	IF2105PM	•2560 × 2048 at 50 Hz •2048 × 2560 at 50 Hz	10-bit

Supported Connectors

Single or Dual-link DVI

Although single-link DVI is only capable of transmitting up to HD (1920 × 1200), our grayscale pixel packing mechanism allows 5 MP (2560 × 2048) images to be sent over single-link DVI.

DisplayPort

- For newer Quadro cards that support DisplayPort output, a DisplayPort-to-DVI adapter is needed as current grayscale monitors only support DVI at this time.
- For DisplayPort-to-single-link DVI conversion, passive adapters such as Hosiden (P/N TYX1602-010307) and Simula (P/N DJ8028B-1000-10E) are tested and recommended.
- To support dual-link resolutions from a DisplayPort connector, an active adapter is required. As shown in Figure 4 this dongle includes a built in USB cable connecting to the USB port providing power to the adapter. NVIDIA recommends the Bizlink DisplayPort-to-DVI-D dual-link cable adapter (P/N KS10014-207).
- The Simula and Bizlink adapters can be purchased from NVIDIA's online at http://store.nvidia.com/ (under the cables category).

Figure 3. DisplayPort to Single-Link DVI Adapter (Passive)

Figure 4. DisplayPort to Dual-Link DVI Adapter (Active)

Grayscale Monitor Settings

When a grayscale compatible monitor is connected to a suitable NVIDIA Quadro solution, the NVIDIA driver automatically detects it and immediately switches to packed pixel mode. Therefore, there are no control panel settings to enable and disable 10-bit grayscale. The only setting required is to enable the grayscale monitor to display at its optimal resolution as shown in the following steps for a 5 MP panel with resolution 2560 × 2048.

1. Open the **Display Properties**.
2. Select the **Settings** tab.
3. Click on **Advanced**.
4. Select the Monitor tab.
5. Uncheck the **Hide modes that this monitor cannot display** check box.
6. Click **Apply**. The maximum resolution is now set to 2560 × 2048.

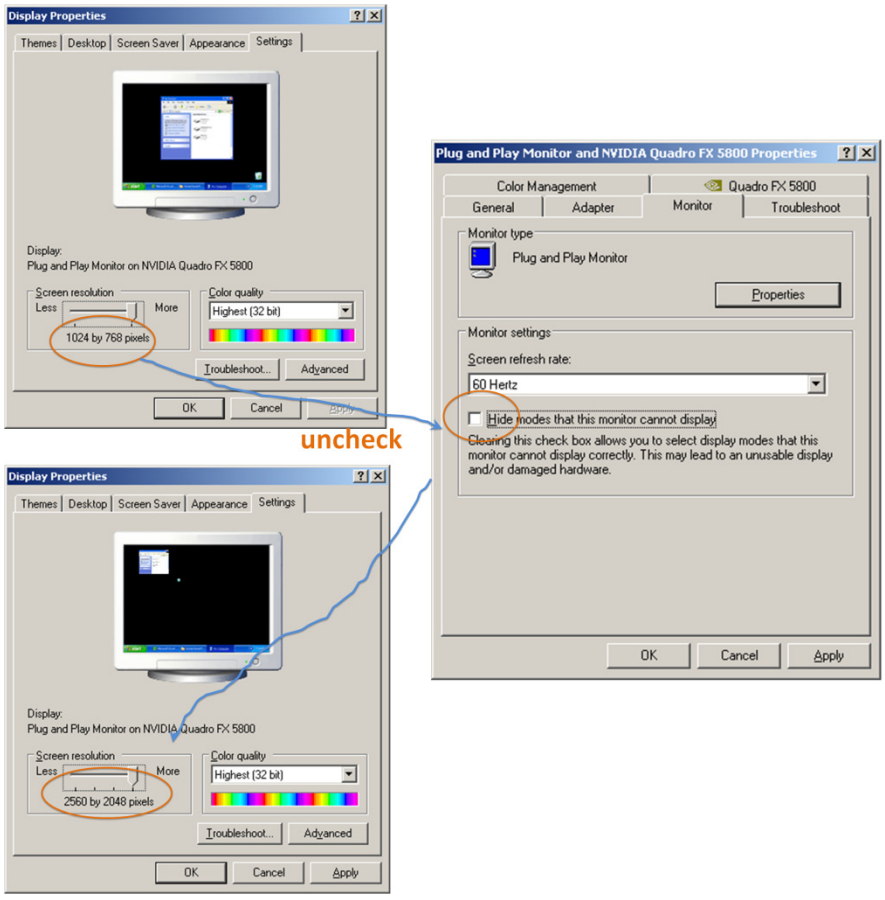

Figure 5. Enable 5 MP Grayscale Monitor to Display Higher Resolution

GRAYSCALE IMPLEMENTATION

Driver Layer

On grayscale enabled Quadro solution, the driver implements a pixel packing mechanism that is transparent to the desktop and to the application. The 24-bit RGB desktop is first converted to 12-bit grayscale using the NTSC color conversion formula and then two 12-bit gray values are packed into 1 RGB DVI pixel and finally shipped to the monitor. This pixel packing allows displaying of 5 MP gray values just using a single-link DVI (that is normally limited to HD resolution).

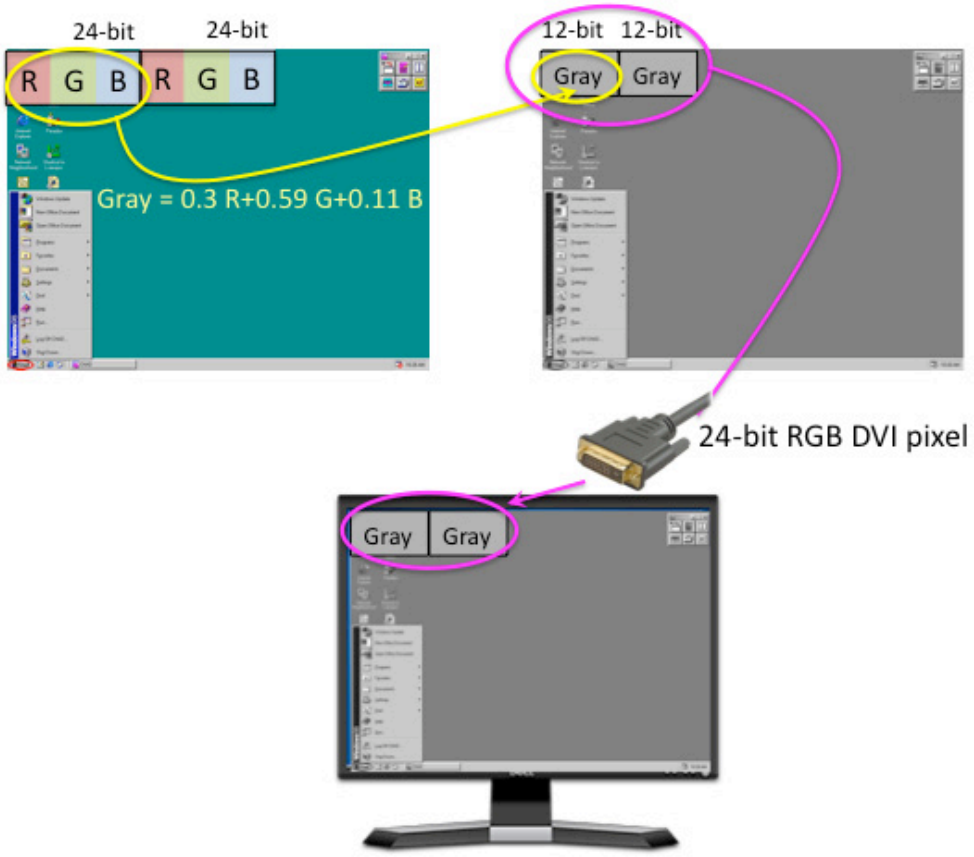

Figure 6. Driver Converts and Packs Desktop from 24-Bit Color to 12-Bit Gray

Application Layer

The 10 and 12-bit grayscale image viewing application is responsible for outputing 24-bit RGB pixels which the driver then converts to 12-bit grayscale values for scanout as described in the previous section.

The application uses a shader that takes in the 12-bit grayscale value from the image and translates it into a 24-bit RGB pixel using a lookup table. The lookup table is generated to find the best RGB pixel with as little as possible differences between the RGB values (preferred is R=G=B) for each grayscale value in the input image. In essence, this process is the inverse of the driver conversion from RGB to grayscale. The end result is that the grayscale image on the desktop looks like a grayscale image on a color monitor.

The integer texture extension, `EXT_texture_integer [4]` in Shader Model 4 is used to store the incoming grayscale image as a 16-bit unsigned integer without converting to floating point representation saving memory footprint by 2×.

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 2);
glTexImage2D(GL_TEXTURE_2D, 0, GL_ALPHA16UI_EXT, width, height, 0, GL_ALPHA_INTEGER_EXT ,
GL_UNSIGNED_SHORT, TextureStorage);
```

The lookup table mapping the grayscale image to 24-bit RGB values is stored as 1D texture. The lookup table dimensions should exactly match the bit depth of the grayscale values expected in incoming image so that no filtering and interpolation operations will be performed thus preserving image precision and fidelity. Changes to contrast, brightness and window level of the image are easily done by changing the lookup table resulting in a 1D texture download without any change to the source image.

```
#extension GL_EXT_gpu_shader4 : enable // for unsigned int support uniform usampler2D
texUnit0; // Gray Image is in tex unit 0
uniform sampler1D  texUnit1; // Lookup Table Texture in tex unit 1
void main(void)
{
        vec2  TexCoord  = vec2(gl_TexCoord[0]);
        //texture fetch of unsigned ints placed in alpha channel
        uvec4 GrayIndex = uvec4(texture2D(texUnit0, TexCoord));
        //low 12 bits taken only
        float GrayFloat = float(float(GrayIndex.a) / 4096.0);
        //fetch right grayscale value out of table
        vec4  Gray      = vec4(texture1D(texUnit1, GrayFloat));
        // write data to the framebuffer
        gl_FragColor  = Gray.rgba;
}
```

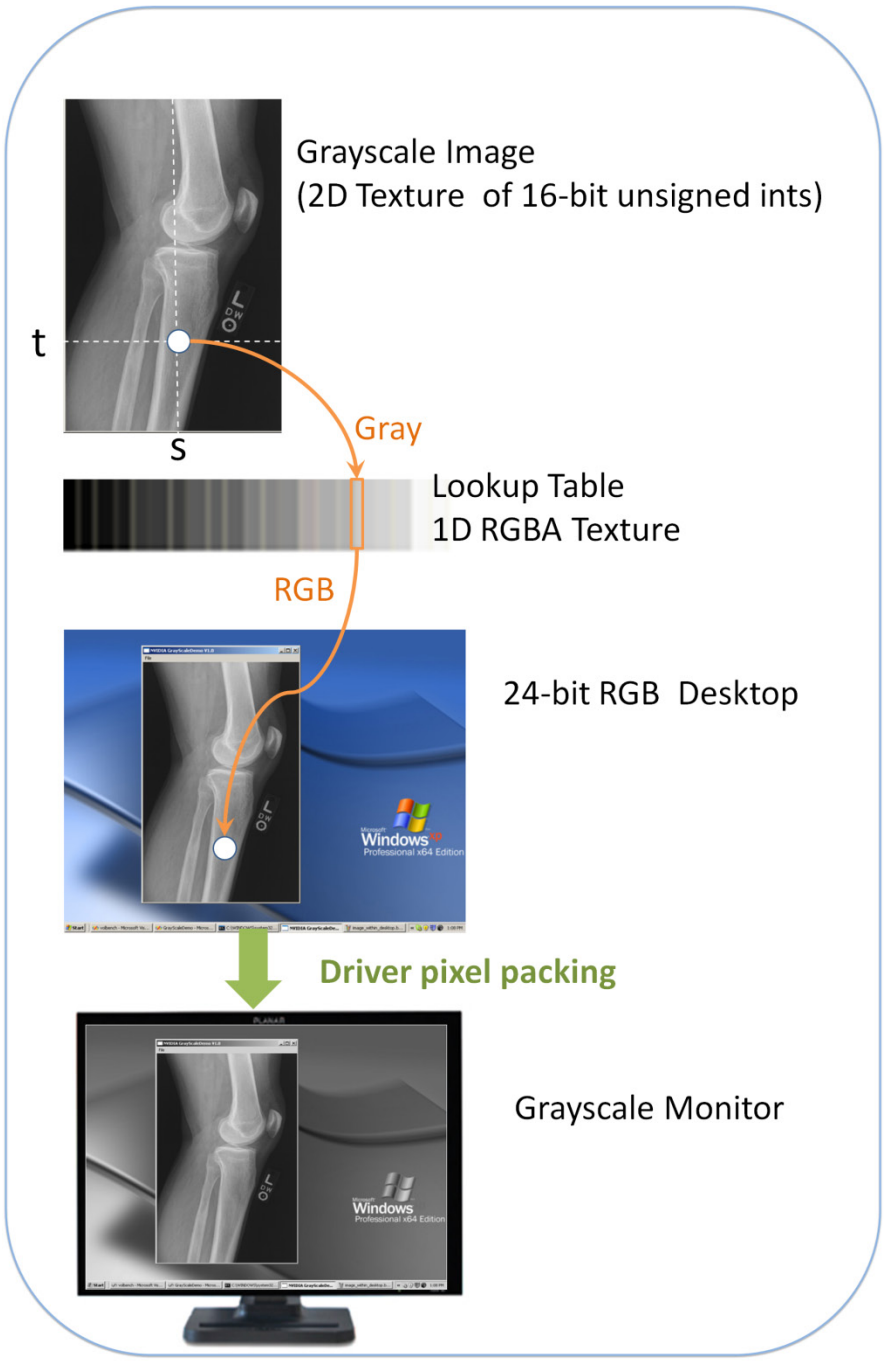

Figure 7. Application Level Texture Setup for 10 and 12-Bit Grayscale Display

MULTI-DISPLAY CONFIGURATIONS

Diagnostic imaging commonly requires multiple displays for side by side modality comparisons. Multi-display configurations are becoming more practical with systems capable of supporting multiple graphics boards that in turn drive multiple displays. A single Quadro board can drive a maximum of 2 displays. Depending on the available PCI slots within a system, multiple cards can be used to drive several displays. These multiple displays can be a mix of regular color LCD panels and specialty grayscale monitors. This section explains the issues that arise from such a heterogeneous configuration and programming pointers to address them. The full source code for the examples is found in the accompanying Grayscale10-bit SDK.

Multi-GPU Compatibility

Grayscale capable Quadro boards can be mixed with other Quadro boards that can drive one or many side displays as shown in Table 3. These “Side Display GPU’s” may not yield the grayscale effect but the system will be compatible. Mixing of GPU’s is only guaranteed to work if the GPU’s are of the same generation.

Table 3. Multi-GPU Compatibility

Side Display Quadro GPU	Grayscale GPU: Quadro 2000D, Quadro 2000, Quadro FX 1800	Grayscale GPU: Quadro 5000, Quadro 4000, Quadro FX 4800, Quadro FX 3800, Quadro FX 4600, Quadro FX 3700	Grayscale GPU: Quadro 6000, Quadro FX 5800, Quadro FX 5600	Grayscale GPU: Quadro Plex 7000, Quadro Plex D2
Quadro 600 Quadro NVS 450 Quadro NVS 420 Quadro NVS 300 Quadro NVS 295	✓	✓	✓	✓
Quadro 2000D Quadro 2000 Quadro FX 1800	✓	✓	✓	✓
Quadro 5000 Quadro 4000 Quadro FX 4800 Quadro FX 3800 Quadro FX 4600 Quadro FX 3700		✓	X	✓
Quadro 6000 Quadro FX 5800 Quadro FX 5600			X	✓

Note: These are theoretical compatibilities assuming the availability of 2 auxiliary power inputs. In practice, the physical system attributes such as availability of PCI slots and their placements will determine the final working set of cards from Table 3. The Quadro FX 5800 and Quadro FX 6000 require the full 2 auxiliary power inputs and therefore only used with lower-end Quadro cards that do not have any auxiliary power requirements.
The mixing of older pre-G80 cards is not supported in grayscale configurations.

Multiple Display Setup

To enable multi-display from the desktop follow these simple steps.

1. Open the **Display Properties**.
2. Select the **Settings** tab.
3. Check the **Extend my Windows desktop onto this monitor** checkbox for each display as shown in Figure 8.

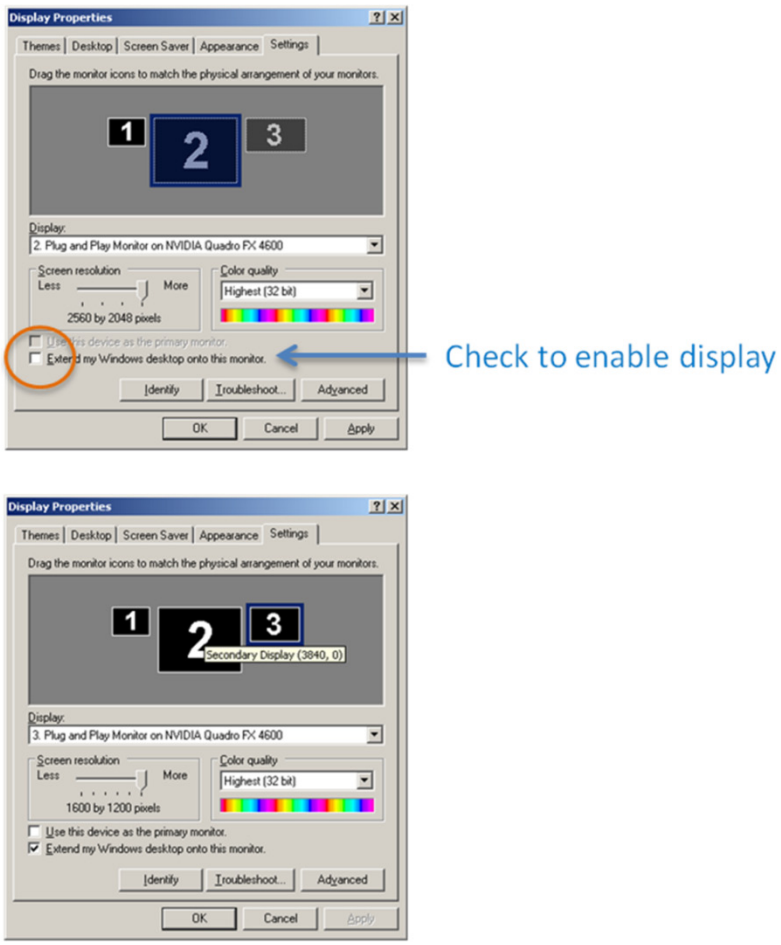

Figure 8. Display Properties Before and After Displays are Enabled

For an application using multiple GPU's and displays it is often useful to programmatically find out their attributes and capabilities. This section and the following ones show code samples to demonstrate that in progressive detail. Following are some data structures used throughout the document examples. The `CDisplayWin` structure defined in `CDisplayWin.[h|cpp]` encapsulates the attributes of each display and the `displayWinList` is a container for all displays. Accessing functions have been omitted to aid readability.

```
class CDisplayWin {
        HWND    hWin; // handle to display window
        HDC             winDC; // DC of display window
        RECT    rect; // rectangle limits of display
        bool    primary; //Is this the primary display
        char    displayName[128]; //name of this display
        char    gpuName[128]; //name of associated GPU
        bool            grayScale; //Is this a grayscale display
public:
        bool    spans(RECT r);//If incoming rect r spans this display

}
#define MAX_NUM_GPUS 4
int displayCount = 0; //number of active displays
//list of displays, each gpu can attach to max 2 displays
CDisplayWin displayWinList[MAX_NUM_GPUS*2];
```

Following is a simple example using the Windows GDI to enumerate the attached displays, gets their extents and also check if the display is set as primary. The following code can be easily modified to include unattached displays.

```
DISPLAY_DEVICE  dispDevice;
DWORD displayCount = 0;
memset((void *)&dispDevice, 0, sizeof(DISPLAY_DEVICE));
dispDevice.cb = sizeof(DISPLAY_DEVICE);
// loop through the displays and print out state
while (EnumDisplayDevices(NULL,displayCount,&dispDevice,0)) {
if (dispDevice.StateFlags & DISPLAY_DEVICE_ATTACHED_TO_DESKTOP) {
        printf("DeviceName    = %s\n", dispDevice.DeviceName);
        printf("DeviceString  = %s\n",dispDevice.DeviceString);
        if (dispDevice.StateFlags &DISPLAY_DEVICE_PRIMARY_DEVICE)
                printf("\tPRIMARY DISPLAY\n");
        DEVMODE devMode;
        memset((void *)&devMode, 0, sizeof(devMode));
        devMode.dmSize = sizeof(devMode);
        EnumDisplaySettings(dispDevice.DeviceName, ENUM_CURRENT_SETTINGS,
                &devMode);
        printf("\tPosition/Size = (%d, %d), %dx%d\n",        devMode.dmPosition.x,
        devMode.dmPosition.y,devMode.dmPelsWidth,    devMode.dmPelsHeight);
        HWND hWin =           createWindow(GetModuleHandle(NULL),devMode.dmPosition.x+50,
                devMode.dmPosition.y+50, devMode.dmPelsWidth-50,     devMode.dmPelsHeight-
50);
        if (hWin) { //got a window
                HDC winDC = GetDC(hWin);
                // TODO - set pixel format, create OpenGL context
        }
        else
                printf("Error creating window \n");
        }//if attached to desktop
        displayCount++;
} //while(enumdisplay);
```

Running this enumeration code on our 3 display example (shown in Figure 8) prints out the following.

```
DeviceName    = \\.\DISPLAY1
DeviceString  = NVIDIA Quadro FX 1800
PRIMARY DISPLAY
Position/Size = (0, 0), 1280x1024

DeviceName    = \\.\DISPLAY2
DeviceString  = NVIDIA Quadro FX 4800
Position/Size = (1280, 0), 2560x2048

DeviceName    = \\.\DISPLAY3
DeviceString  = NVIDIA Quadro FX 4800
Position/Size = (3840, 0), 1600x1200
```

Note: The enumeration shown in this section abstracts special hardware capabilities of the displays such as grayscale or color capability. For such physical display details, we need to access to the Extended display identification data (EDID) the data structure provided by the computer display to the graphics card. This is described in the next section.

Mixing Grayscale and Color Displays

The previous section demonstrated how to get the general characteristics of a display such as extent etc, but more specific properties of monitors will decide how to layout our application. For example, user interface and launching elements are normally placed on the regular color LCD's while the radiological images will be rendered to the grayscale displays. A display is defined to be grayscale compatible if both the monitor and the GPU attached are grayscale enabled. To determine if a monitor is grayscale we parse its EDID to get the model name and compare it with the list of enabled monitors. This EDID is provided by the NVIDIA NVAPI [5] – an SDK that gives low level direct access to NVIDIA GPUs and drivers on all windows platforms. The following example shows enumerating the attached displays and its associated panel and GPU string. Refer to the complete source in `CheckGrayscale.cpp` for error checking functions and the `isGrayscaleGPU` and `isGrayscaleMonitor` string parsing functions.

```
// Declare array of displays and associated grayscale flag
NvDisplayHandle hDisplay[NVAPI_MAX_DISPLAYS] = {0};
NvU32 displayCount = 0;
// Enumerate all the display handles
for(int i=0,nvapiStatus=NVAPI_OK; nvapiStatus == NVAPI_OK; i++) {
nvapiStatus = NvAPI_EnumNvidiaDisplayHandle(i, &hDisplay[i]);
if (nvapiStatus == NVAPI_OK) displayCount++;
}
printf("No of displays = %u\n",displayCount);

//Loop through each display to check if its grayscale compatible
for(unsigned int i=0; i<displayCount; i++) {
//Get the GPU that drives this display
NvPhysicalGpuHandle hGPU[NVAPI_MAX_PHYSICAL_GPUS] = {0};
NvU32 gpuCount = 0;
nvapiStatus =
NvAPI_GetPhysicalGPUsFromDisplay(hDisplay[i],hGPU,&gpuCount);
nvapiCheckError(nvapiStatus);



//Get the GPU's name as a string
NvAPI_ShortString gpuName;
NvAPI_GPU_GetFullName (hGPU[0], gpuName);
printf("Display %d, GPU %s",i,gpuName);
nvapiCheckError(nvapiStatus);

//Get the display ID for subsequent EDID call
NvU32 id;
nvapiStatus = NvAPI_GetAssociatedDisplayOutputId(hDisplay[i],&id);
nvapiCheckError(nvapiStatus);

//Get the EDID for this display
NV_EDID curDisplayEdid = {0};
curDisplayEdid.version = NV_EDID_VER;
nvapiStatus = NvAPI_GPU_GetEDID(hGPU[0],id,&curDisplayEdid);
nvapiCheckError(nvapiStatus);

//Check if the GPU & monitor both support grayscale
//and set the grayFlags table
if (isGrayscaleGPU(gpuName)&& \\
        isGrayscaleMonitor(curDisplayEdid.EDID_Data,NV_EDID_DATA_SIZE))
        displayWinList[i].grayScale = true;
else
        displayWinList[i].grayScale = false;
}
```

Directed GPU Rendering

In a multi-GPU setup, the default behavior is for OpenGL commands to be sent to all GPUs. While this works for many applications, performance is gated by the capabilities of the lowest-end card. In a typical grayscale setup, the side display with GUI elements is normally connected to a lower-end Quadro while the grayscale panels are connected to a higher-end Quadro card. It is desirable to limit grayscale rendering to the GPUs that are driving the grayscale panels and not involve the side GPU at all in the render process. Previous approaches required programmatically selecting the GPU using OpenGL extensions which can quickly become an additional programming burden for a radiology developer. The newer Quadro drivers have a feature called “Directed Rendering” that allows the user to target the GPU for rendering and decouple it from the display GPU. This is done via the NVIDIA Control Panel as shown in Figure 9 or programmatically using NVAPI (see accompanying SDK). When the render GPU and the display GPU are different, the driver transparently uses a fast transfer path on Quadro cards to split the rendered images onscreen. By default, the driver will pick the biggest GPU for render.

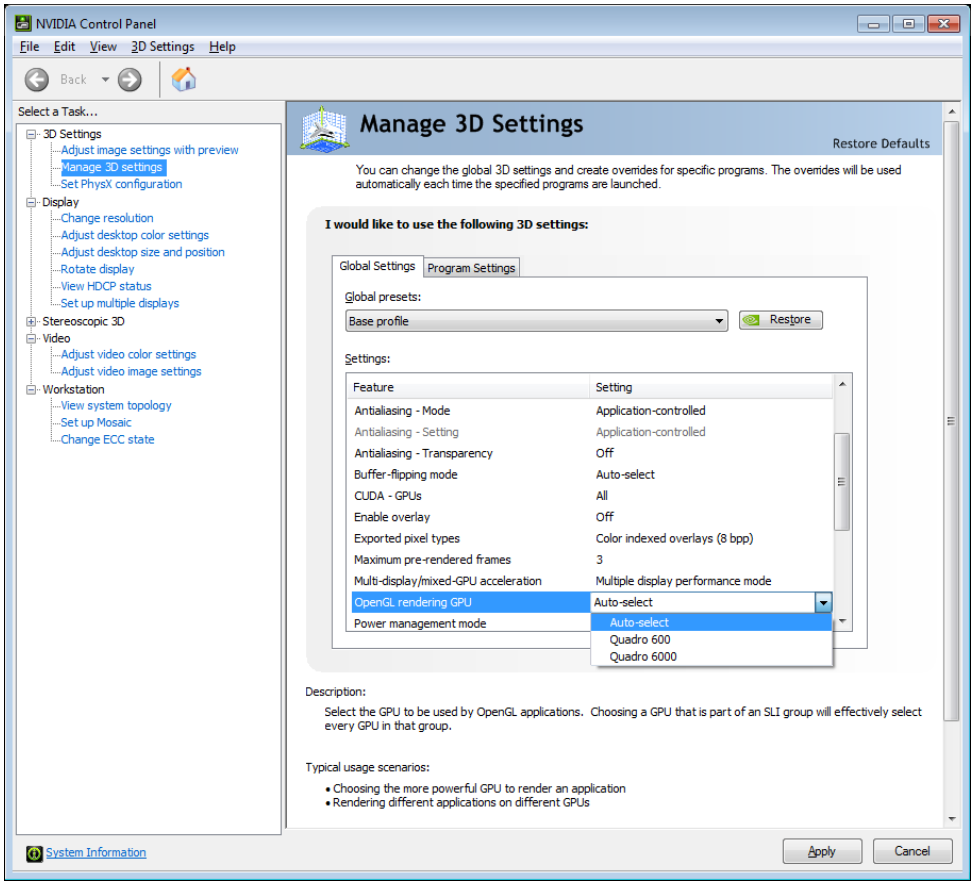

Figure 9. Setting Render GPU from NVIDIA Control Panel

TYPICAL MULTI-DISPLAY CONFIGURATIONS

We examine the commonly used multi-display setups that mix grayscale monitors and color panels and their underlying GPU configuration.

Case 1. Two 5 MP Grayscale Displays Driven by One GPU

The most commonly used configuration for diagnostic imaging, a high-end Quadro GPU drives 2 5 MP grayscale displays. One or two side displays are driven by a low-end NVS™ card (if there are no PCI Express ×16 slots available) or another Quadro card.

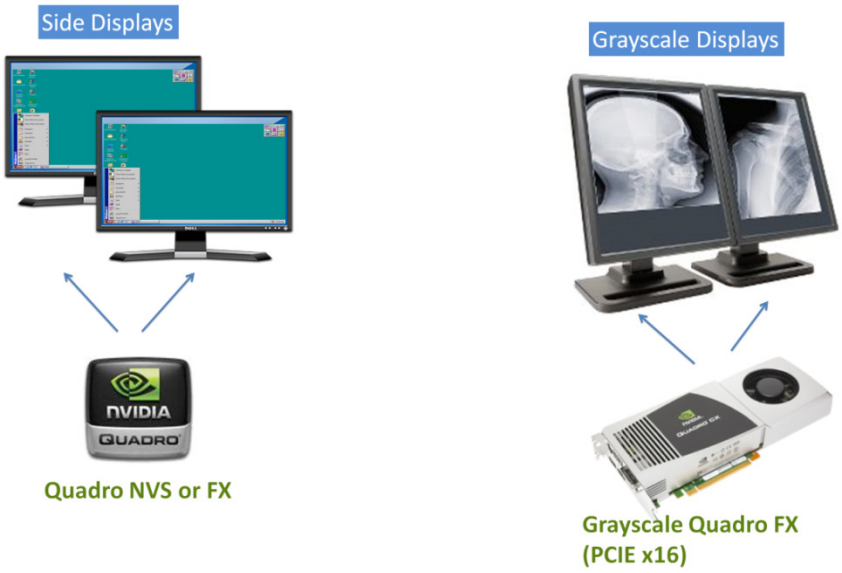

Figure 10. 10 MP Grayscale Display Configuration

Table 4. Characteristics for 10 MP Setup

Total Resolution	10 MP	5120 × 2048 (landscape) or 4096 × 2560 (portrait)
Side Display (Primary)	Quadro NVS 300 Quadro NVS 295 Quadro NVS 420	PCIe ×1 option; good for systems with only 1 avalable PCIe ×16 slot that is used for the grayscale GPU
	Quadro NVS 450 Mid, high-end GPUs	Occupies 1 PCIe ×16 slot; recommended for systems with at least 2 PCIe ×16 slots
	Quadro FX 4600 Quadro FX 4800	Physically spans 2 PCIe slots; recommended for systems with at least 2 PCIe ×16 slots
Grayscale Display	Grayscale GPUs (Table 3)	Requires 1 or 2 PCIe slots depending on the GPU

Case 2. Four 5 MP Grayscale Displays Driven by Two GPUs

Two grayscale capable Quadro GPUs drive four 5 MP grayscale displays. One or two side displays are driven by a low-end Quadro NVS card or a mid to high-end Quadro FX card depending on the available PCI Express ×16 slots in the system.

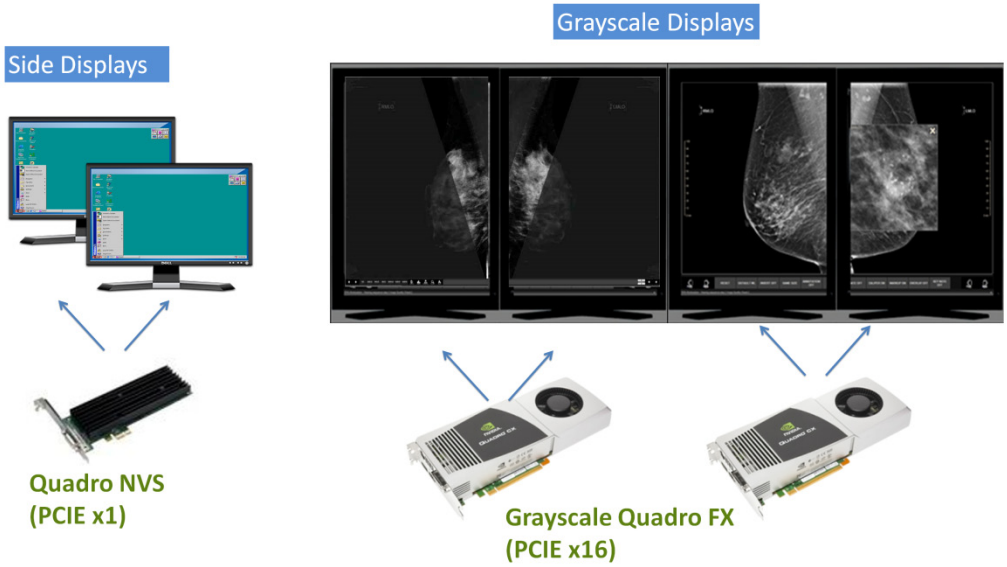

Figure 11. Three GPUs Driving a 20 MP Grayscale Display

Table 5. Characteristics for the 20 MP Setup

Total Resolution	20 MP	10, 240 × 2048 (landscape) or 8192 × 2560 (portrait)
Side Display (Primary)	•Quadro NVS 300 •Quadro NVS 295 •Quadro NVS 420 •Mid high-end grayscale GPU	•1 PCI Express ×1 slot •1 PCI Express ×16 slot
	Quadro NVS 450 Mid, high-end GPUs	Occupies 1 PCIe ×16 slot; recommended for systems with at least 2 PCIe ×16 slots
Grayscale Display GPU 1	Grayscale GPUs (Table 3)	1 PCI Express ×16 slot. May occupy 2 PCIe slots
Grayscale Display GPU 2	Grayscale GPUs (Table 3)	1 PCI Express ×16 slot. May occupy 2 PCIe slots

REFERENCES

[1] Digital Imaging and Communications in Medicine (DICOM)- Part 14 grayscale standard display function. http://medical.nema.org

[2] NDS Dome E5 Display
http://www.ndssi.com/products/dome/ex-grayscale/e5.html

[3] Eizo Radiforce GS520 Display
http://www.radiforce.com/en/products/mono-gs520-dm.html

[4] Integer Texture Extension
http://www.opengl.org/registry/specs/EXT/texture_integer.txt

[5] NVIDIA NVAPI – www.nvapi.com

[6] Ian Williams, HD is now 8MP &HDR, Slides from NVISION 2008.
http://www.nvidia.com/content/nvision2008/tech_presentations/Professional_Visualization/NVISION08-8MP_HDR.pdf

IMPLEMENTATION DETAILS

The accompanying source code is divided into 3 separate projects. The intent is for these components to be mixed and matched according to the user application requirements.

- **GrayscaleDemo.sln**
  - **GrayscaleDemo.[cpp|h]** – An example demo application that does the various texture setups and allows the user to choose a grayscale image for display.
- **CheckGrayscale.sln**
  - **CDisplayWin.[cpp|h]** – Class CDisplayWin that encapsulates all attributes of an attached display such name, extents, driving GPU, etc.
  - **CheckGrayscale.cpp** – Main program that enumerates all attached GPUs and displays using Win GDI API and uses NVIDIA NVAPI to check the displays that are grayscale compatible.
- **DirectedRendering.sln**
  - **DirectedRendering.cpp** – Main program that shows targeting a specific GPU for rendering using NVAPI.

Notice

HDMI

ROVI Compliance Statement

OpenCL

Trademarks

Copyright