AMD's 10-bit Video Output Technology

Introduction

Display devices with a greater bit depth than the conventional 8-bits per color channel are rapidly gaining popularity in application areas such as medical imaging, professional photography, graphics design, movie production, and entertainment in general. What makes these devices attractive to a wide array of fields is their ability to represent colors with a greater fidelity than it is possible with conventional display devices. This means what we see in the real world more accurately matches what we see on the screen, ensuring that design and diagnosis errors due to monitor limitations are minimized.

In order to benefit from the increased bit depth of these display devices, graphics cards used to drive them should be capable of outputting higher bit depth information as well. For example, if a display device supports 10-bits per color channel input, its capacity will not be fully exploited unless the graphics card outputs a matching bit depth. This is where AMD's most recent series of workstation graphics cards come into play by natively supporting the emerging standard for 10-bits per color channel (or 30-bits per pixel) video output.

In this paper, we first emphasize the necessity of higher bit depths for color critical tasks followed by an overview of the 10-bit monitors currently available. We then demonstrate how users can easily benefit from this technology using AMD's graphics cards capable of 10 bit output.

The Need for 10-bit Displays

Conventional display devices use 8-bits per color channel (or 24-bits per pixel) to display images and video. Although this amounts to more than 16 million colors, it still corresponds to a fraction of the colors we perceive in the real-world. This is illustrated in Figure 1, where the green triangle shows the boundaries of the sRGB[1] color space on the CIE-xy chromaticity diagram. Conventional sRGB compliant 8-bits monitors can only represent the colors that lie in this triangle, while the human eye is capable of perceiving all the colors in the entire chromaticity diagram. This discrepancy is further emphasized by the fact that today's most professional cameras and printers have a color gamut larger than that of sRGB (such as Adobe RGB[2] shown by the red triangle in Figure 1), creating a bottleneck on the display side. Figure 2 demonstrates this problem where the blue areas in a photograph taken with the Adobe RGB setting of a camera cannot be accurately represented on a conventional monitor.

To solve this problem, display devices can operate in a color space with a larger gamut than that of the sRGB. This can be achieved by using purer (more saturated or monochromatic) primaries to stretch the

[1] sRGB is a standard color space proposed by HP and Microsoft to allow for consistent color reproduction on typical monitors, printers and the Internet. It has the same primary set as the HDTV also known as ITU-R BT. 709.
[2] Adobe RGB is an RGB color space with a larger gamut than sRGB. It is developed by Adobe with the printing industry in mind, as there were many colors that can be printed but not displayed by a smaller gamut color space.

corners of the color gamut toward the edges of the chromaticity diagram. However, when the color gamut is expanded, using only 8-bits per color channel ceases to be sufficient to display a smooth image without banding artifacts. This is illustrated in Figure 3 with a sample set of colors in both sRGB and Adobe RGB color spaces. Note that, the spacing between the color samples increases in the Adobe RGB space since it occupies a larger area. Another example is shown in Figure 4, where only 6-bits are used to cover what should normally be an 8-bit range. Similar problems will occur if 8-bits are used to cover what should normally be a 10-bit, that is, a larger range. Therefore to convey a smooth image, one needs to use additional shades of colors, hence the need for 10-bit displays.

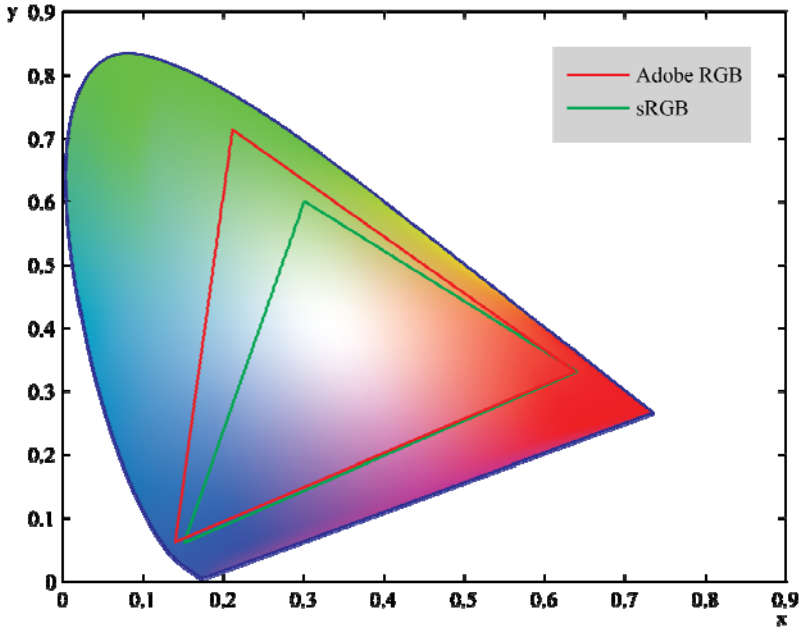

Figure 1: The 2D color gamut of the sRGB and Adobe RGB color spaces are superimposed on the CIE-xy chromaticity diagram. Note that the colors shown inside the diagram are for illustration purposes.

Figure 2: A photograph taken with an Adobe RGB camera may not be accurately represented on an 8-bit sRGB monitor. The blue areas show the out-of-gamut colors.

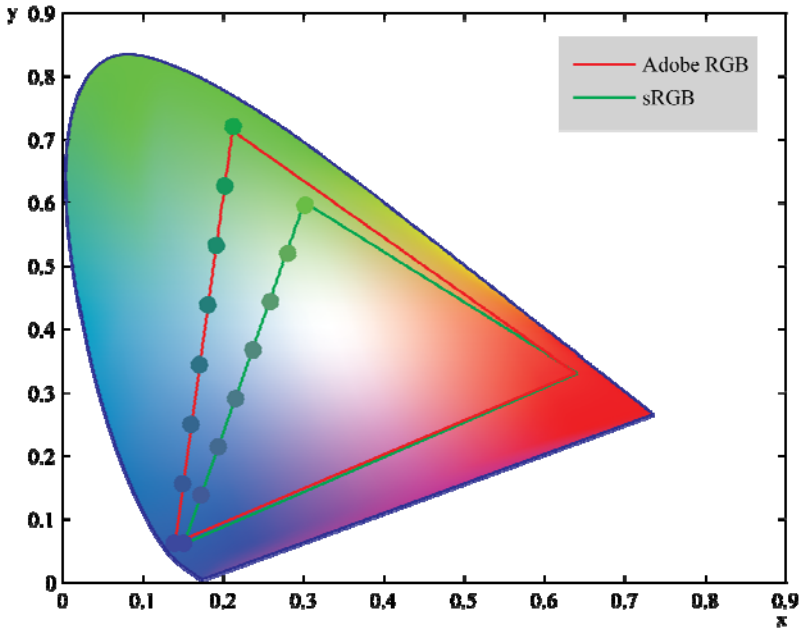

Figure 3: When the same number of colors are used to cover a larger color space, the distance between them increases which may lead to banding artifacts. To avoid this problem, the bit depth of the display device needs to be increased.

Figure 4: This figure illustrates the banding artifacts that would occur if a smaller number of bits are used to cover a larger color range. For illustration purposes 6-bit are used to cover a subset of the 0-255 range – similar artifacts will occur if 8-bits are used to cover a larger color space such as the Adobe RGB.

10-bit Display Devices

10-bit displays use 10 bits to represent each of the red, green and blue color channels. In other words, each pixel is represented with 30 bits, instead of the 24 bits used in conventional displays. To this end, 10-bit displays can generate more than a billion shades of colors.

In medical imaging, display devices with higher bit depths have already been in use as they allow for more accurate identification and diagnosis. Two examples of medical displays that support 10-bits are *Barco Coronis 5MP Mammo* and *Eizo GS510*. Both are grayscale displays and are commonly used in mammography applications. There exist full color 10-bit medical displays as well such as the *Eizo RadiForce R31*, which can be used to visualize MRI, PET, and CR scans.

Higher bit depth display devices are also entering the mainstream in graphics design, professional photography, compositing, and cinema post production. Recently HP unveiled *DreamColor LP2480zx*, which is a 24" LED backlit LCD display capable of presenting 30-bits per pixel color information. Another 30-bit display was introduced earlier by NEC (*LCD2180WG-LED*), which also uses LEDs to back light the LCD panel. Both of these displays prefer LED backlighting instead of the more traditional Cold Cathode Fluorescent Lamps (CCFL), since LEDs offer a much narrower color spectrum than fluorescents, enabling more saturated primary colors. Both the HP and the NEC display are capable of accurately representing the entire Adobe RGB color gamut.

AMD's 10-bit Display Support

AMD's most recent series of workstation graphics cards give full support to various types of 10-bit displays including all of the models mentioned above. The communication interface can currently be single link DVI, dual link DVI, and DisplayPort depending on the input/output ports configuration of the monitor and the graphics card. Through a proprietary packed pixel format, very high resolution grayscale medical displays can be natively supported through a single link DVI. Figure 5 demonstrates how the user can enable the 10-bit display support in the OpenGL driver by simply checking the relevant checkbox in the Catalyst™ Control Center (CCC).

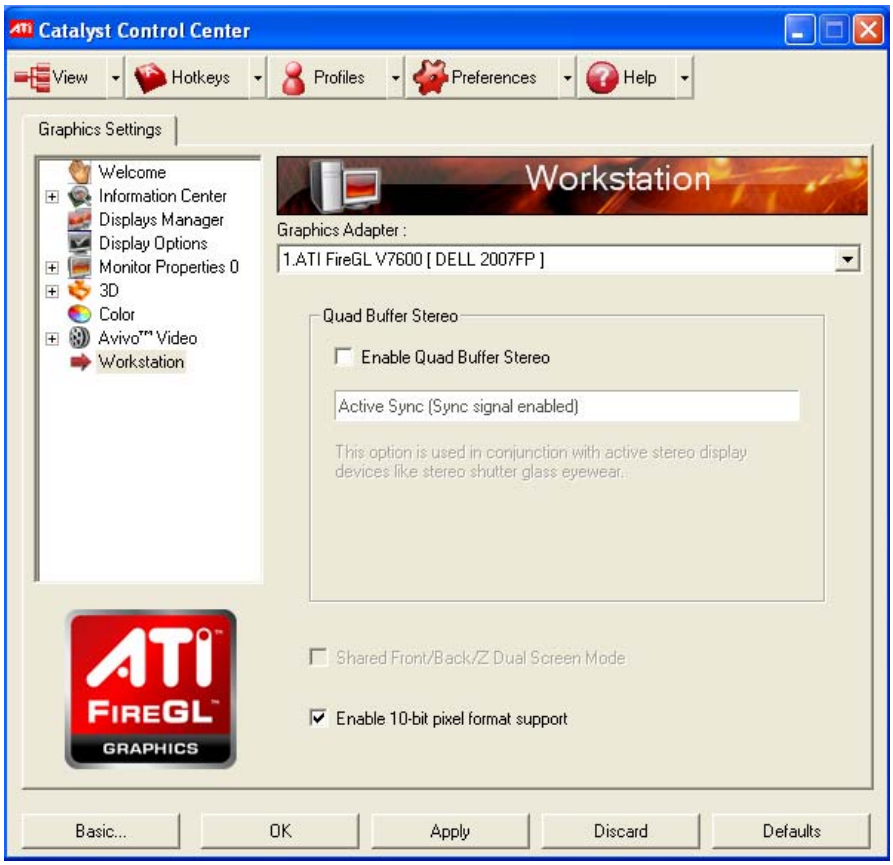

Figure 5: 10-bit display feature can be enabled by checking the "Enable 10-bit pixel format support" checkbox in the Catalyst Control Center. Note that this feature is only available in workstation (ATI FireGL™) cards.

Once the 10-bit pixel format support is enabled the system will request a reboot and after that any 10-bit aware OpenGL application will be displayed in 10-bits without any clamping. In the following, we demonstrate how an application programmer can easily create a 10-bit aware OpenGL application.

Choosing a 10-bit Pixel Format

The first step in creating a 10-bit OpenGL application is choosing an appropriate pixel format. Depending on the windowing library used, this can be achieved in slightly different ways. We first demonstrate the GLUT method followed by the WGL_ARB_pixel_format extension approach.

GLUT

To request a 10-bit pixel format in GLUT, the user needs to specify the appropriate display string before creating an OpenGL window using:

```
// request 10 bit colors, a depth buffer, and double buffering
glutInitDisplayString("red=10 green=10 blue=10 depth double");
```

After this call, any OpenGL window created with `glutCreateWindow()` will have a 10-bit per color channel pixel format (the window does not have to be in fullscreen). This can be verified by querying the number of bits for each color channel:

```
// verify that each color channel has 10 bits
```

```
int red_bits, green_bits, blue_bits;
glGetIntegerv(GL_RED_BITS, &red_bits);
glGetIntegerv(GL_GREEN_BITS, &green_bits);
glGetIntegerv(GL_BLUE_BITS, &blue_bits);
```

WGL_ARB_pixel_format Extension

A 10-bit pixel format can also be requested using the `wglChoosePixelFormatARB()`function provided by the WGL_ARB_pixel_format extension as shown below:

```
HDC device_context; // set previous to this code
int attribs[64] = {
	WGL_SUPPORT_OPENGL_ARB, TRUE,
	WGL_DRAW_TO_WINDOW_ARB, TRUE,
	WGL_PIXEL_TYPE_ARB, WGL_TYPE_RGBA_ARB,
	WGL_RED_BITS_ARB, 10,
	WGL_GREEN_BITS_ARB, 10,
	WGL_BLUE_BITS_ARB, 10,
	WGL_DEPTH_BITS_ARB, 24,
	WGL_DOUBLE_BUFFER_ARB, TRUE,
	0, // zero terminates the list
};

static float fattribs[64] = {
	0.0f, // zero terminates the list
};

const int format_max = 256;
int formats[format_max];
unsigned int format_count;
wglChoosePixelFormatARB(device_context, attribs, fattribs,
                        format_max, formats, &format_count);

// "format_count" suitable formats are in "formats".
if(format_count == 0) {
	// there were no suitable formats
} else {
	// For this example we just use the first match.
	result = SetPixelFormat(device_context, formats[0], &pfd);
	if(result == 0) {
		// error can be retrieved from GetLastError()
	}
}
```

Creating a 10-bit Texture

The previous section highlighted several methods to choose a 10-bit pixel format. It is important to note that once a 10-bit pixel format is chosen any smooth shading operation will immediately take advantage of the increased bit depth. In other words, the user does not need to explicitly provide 10-bit input data to benefit from the increased precision. This is due to the fact the internal processing of colors in the graphics card is in floating-point precision with 10-bit (or 8-bit) conversion occurring only at the output stage.

However, it is also possible to explicitly create and work with 10-bit textures as will be explained in this section. 10-bit texture support is exposed to the user through a packed `RGB10_A2` texture format, which contains 10 bits for each of the color channels and 2 bits for the alpha component. For convenience, we can start by defining an auxiliary union which can be used to set the values of a 10 bit texture:

```
// Auxiliary union used to set the values of the 10 bit texture
union format2_10_10_10
{
    struct {
        unsigned int red  :10;
        unsigned int green:10;
        unsigned int blue :10;
        unsigned int alpha:2;
    } bits;
    unsigned int u32All;
};
```

Next we allocate a memory buffer which will be filled with the user data:

```
// Allocate a buffer which will be used as the 10 bit texture
int drawWidth, drawHeight;
format2_10_10_10* drawData = new format2_10_10_10 [drawWidth * drawHeight];
```

After the buffer is successfully created, it can be filled with user data coming from any 10-bit source such as a DICOM image or a RAW camera output:

```
// Fill the buffer with values from the user. userRed, userGreen,
// and userBlue are unsigned integers in the range [0,1024].
format2_10_10_10 aux;
for (int y = 0; y < drawHeight; ++y)
  for (int x = 0; x < drawWidth; ++x)
  {
      aux.bits.red   = userRed;   // user input
      aux.bits.green = userGreen; // user input
      aux.bits.blue  = userBlue;  // user input

      drawData[x + y * drawWidth].u32All = aux.u32All;
  }
```

All that remains to be done is to create a 2D texture, specify the buffer we just filled in as the texture data source, and set the desired texture parameters:

```
// Create and bind a new texture
GLuint texID;
glGenTextures(1, &texID);
glBindTexture(GL_TEXTURE_2D, texID);

// Fill the texture image from the 10 bit buffer data
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB10_A2, drawWidth, drawHeight,
             0, GL_RGBA, GL_UNSIGNED_INT_2_10_10_10_REV, drawData);

// Set desired texture parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

Finally, we map the texture on a screen size quad for display on the screen:

```
// Map the texture on a screen size quad for display
glEnable(GL_TEXTURE_2D);
glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0);
    glVertex2f(0.0, 0.0);
```

```
    glTexCoord2f(0.0, 1.0);
    glVertex2f(0.0, drawHeight);

    glTexCoord2f(1.0, 1.0);
    glVertex2f(drawWidth, drawHeight);

    glTexCoord2f(1.0, 0.0);
    glVertex2f(drawWidth, 0.0);
glEnd();
```

This sample code demonstrated how to create and display a 10-bit per color channel texture with input data explicitly specified by the user. Additionally, for off-screen rendering, 10-bit textures can be attached as render targets (i.e. color attachments) to framebuffer objects.

Conclusions

As a result of the enhanced presentation quality enabled by higher bit depths, 10-bit displays are rapidly becoming norm rather than exception in application areas such as medical imaging, professional photography, graphics design, compositing, and post production. To realize their full potential, these displays demand a higher bit depth video output from the graphics cards that are used to drive them. AMD's most recent series of workstation cards are designed to meet this demand by natively supporting 10-bits, ensuring that crucial information is never lost in any stage of the rendering and display pipeline.

Further Reading

R. W. G. Hunt. *The Reproduction of Colour*, Sixth edition. Chichester, UK: John Wiley and Sons Ltd., 2004.

C. Poynton. *Digital Video and HDTV: Algorithms and Interfaces.* San Francisco: Morgan Kaufmann Publishers, 2003.

E. Reinhard, E. A. Khan, A. O. Akyüz, and G. M. Johnson. *Color Imaging: Fundamentals and Applications*. Wellesley: AK Peters Ltd., 2008.

G. Wyszecki and W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae,* Second edition. John Wiley and Sons Ltd., 2000.