Freescale Semiconductor
Application Note

Document Number: AN5072
Rev 0, 02/2015

Introduction to Embedded Graphics with Freescale Devices

by: Luis Olea and Ioseph Martinez

Contents

1 Introduction

The purpose of this application note is to explain the basic concepts and requirements of embedded graphics applications, helping software and hardware engineers easily understand the fundamental concepts behind color display technology, which is being expected by consumers more and more in varied applications from automotive to industrial markets.

This section helps the reader to familiarize with some of the most used concepts when working with graphics. Knowing these definitions is basic for any developer that wants to get started with embedded graphics systems, and these concepts apply to any system that handles graphics irrespective of the hardware used.

1.1 Graphics basic concepts

Pixels

A pixel (short for picture element) is a physical point in a raster image. Depending on the format of the image, a pixel may take up from just 1 bit of memory to 32 bits each. The space in memory each bit takes is called “bits per pixel” and it tells how many different colors can each pixel be.

RGB

The RGB color model is an additive color model in which red, green, and blue light are added together in various percentages to reproduce a broad array of colors. The main purpose of the RGB color model is for sensing, representation, and display of images in electronic systems.

RGB is one of the most common and simple image types, in which a number of bits are assigned to each of the color components, and each pixel in the image stored requires the sum of the bits assigned to its red, green, and blue subpixels.

The basic idea behind representing colors with these three components (red, green, and blue) is that superimposing (adding) the light of these three colors in different percentages can be used to reproduce a very wide range of visible colors.

Figure 1. Combinations of red, green, and blue light

Resolution

The resolution of a display is the number of distinct pixels in each direction (horizontal and vertical) that can be displayed. It is usually expressed in “WIDTH x HEIGHT” format.

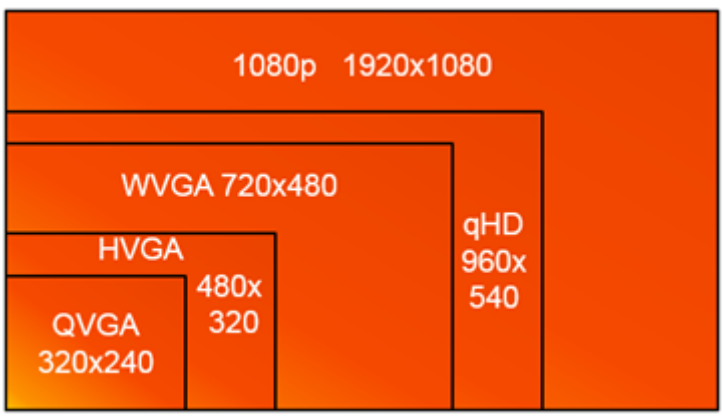
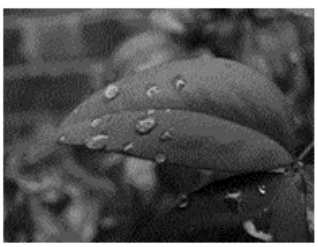
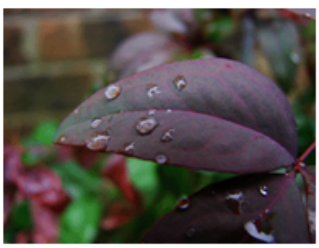
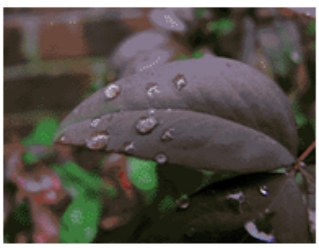


Figure 2. Common display resolutions

Bits per pixel

BPP, also known as color depth or bit depth, is the number of bits that are used to indicate the color of a single pixel in a bitmapped image or video frame buffer. The selection of the proper BPP for images in an embedded system is critical since it allows the designer to select a quality vs resource tradeoff. Higher BPP images look better, but may be more CPU/GPU intensive and take up more RAM and FLASH storage. Low BPP images may look worse but save valuable resources.

Table 1. Comparison of different color depths

Image	BPP	Image	BPP
	1 bit per pixel 300 x 225 pixels 8438 bytes 8.2 KB		8 bits per pixel 300 x 225 pixels 67500 bytes 65.92 KB
	2 bits per pixel 300 x 225 pixels 16875 bytes 16.4 KB		16 bits per pixel 300 x 225 pixels 135000 bytes 131.84 KB
	4 bits per pixel 300 x 225 pixels 33750 bytes 32.9 KB		24 bits per pixel (true color) 300 x 225 pixels 202500 bytes 197.75 KB

The selection of the proper bits per pixel for images is apparent in Table 1. Low BPP allow using less memory, but the selection of colors is poor. Ideally, the designer will choose the lowest BPP possible that does not have noticeable quality loss.

Alpha (transparency)

Alpha or transparency is another component of a pixel. It indicates how transparent is that pixel. Some image formats may or may not include it, and its width (bit depth) can vary. For example, images with a bit depth of 32 bpp have an 8-bit alpha channel. Figure 3 shows a 32 bpp image where the transparent color is represented by a gray checker box background. This is typically done on picture editing tools to make easier to identify transparent pixels.

Figure 3. 32 bpp image with transparent background

Blitting and blending

Blitting means combining at least two bitmaps through a raster operation. In early graphics systems that did not have hardware acceleration, raster operations were very costly as they have to be executed on each of the pixels that overlap, that is, the two bitmaps that were to be combined. So raster operations were kept simple, such as an inexpensive logic operation (XOR, OR, etc).

In current graphics systems, blending allows combining two (or more) images in which we are able to define the ponderation. This means that the front image can have a certain transparency percentage (arbitrary, we can pick it to be from 0% to 100%). The image behind it will have the complement; if we defined the image in the foreground to be 80% transparent, the background image will be 20% transparent. The resulting operation is defined by following equation:

$$C_0 = C_a\alpha_a + C_b\alpha_b(1-\alpha_a)$$

Equation 1. Alpha blending operation

In the above equation, C_a represents a channel (red, green, or blue) and α_a represents the alpha or transparency of the image. After a simple inspection of the equation it is evident that it is just a pondered sum of two pixel channels: the foreground channel C_a has a transparency of α_a so the background image channel must have the complementary transparency of $(1 - \alpha_a)$.

This operation must be performed once per channel (red, green, and blue) and repeated for as much images we want to overlap/combine. It is evident that two additions and three multiplications per pixel (where we could have areas of 1024 x 768 pixels or more) would be a heavy load for the CPU, hence the use of hardware acceleration for graphics systems.

The Equation 1 was introduced by Porter and Duff on a paper published on 1984. This paper includes other useful and common blending operators for 2D images operations.

Coordinate system

Each point in the screen can be addressed using a coordinate pair, which represent the x and the y axis coordinates. Contrary to the cartesian coordinate system, the coordinate system in graphics usually start with coordinate (0,0) on the upper left corner of the screen or framebuffer. The Y axis is positive going down the screen, and the positive X axis is to the right of the screen. In 3D graphics, there is a Z axis or coordinate, which is positive going out of the screen toward the viewer.

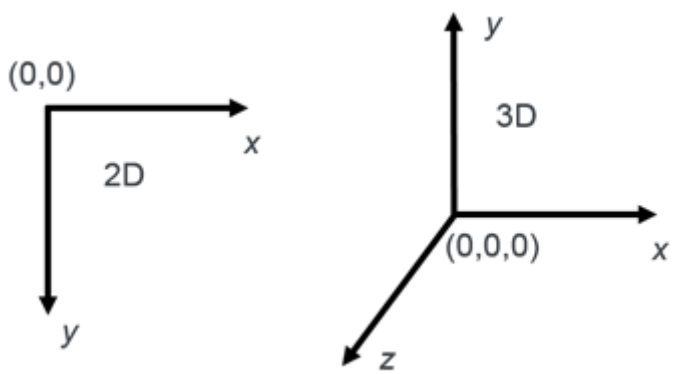


Figure 4. 2D and 3D coordinate systems

Stride

The stride of an image refers to the number of locations in memory between the start of successive image lines. In other words, the stride is the number of bytes one has to move to get from the last pixel of a line in the image to the first pixel of the next horizontal line in the image. Stride cannot be smaller than the pixel size (because it would indicate that the pixels overlap) but it can be larger, indicating that there is extra space or padding between horizontal lines of an image.

Image Compression

Images can be compressed or uncompressed which give different advantages and disadvantages.

- **Uncompressed**

  Images require the most storage space but the advantage is they usually don't require any extra processing to be displayed or manipulated into the frame buffer or display.

- **Compressed**

Compressed images are stored with some degree of size reduction. Although this saves permanent storage space (i.e. flash), the image has to be decompressed when it needs to be displayed, taking CPU and RAM overhead. The 2D_ACE (also known as DCU) has hardware RLE decompression modules which can get rid of both of these disadvantages.

- *Lossy:* compress the data in the image to a point that some information is lost (quality is reduced), but storage space is saved at the expense of quality.
  - JPEG: Joint Picture Experts Group. A standard for compressing images that uses complex techniques involving two dimensional discrete cosine transforms

  Lossless: it stores the image in such a way that it occupies less storage space than an uncompressed image, but no information is lost: the original image can be reproduced exactly as it was before compression. Some examples:-
  - Color Palette. Stores the finite set of colors in the image in a color look up table. The pixels are stored as indexes pointing to this table.
  - RLE: Run Length Encoding. Compresses the image by using a notation which takes advantage of series of repeated patterns in the image data.

Color Palette

An image that is stored with a color palette consists of an array of indexes and a color palette or color lookup table (CLUT). The concept is that an image does not need the entire 24-bit range of true colors. For example, if only 256 different colors are used, we store a table with these 256 colors in a CLUT, and the image pixels contain just an 8-bit index that points to a color in the CLUT, instead of storing a 24-bit value for each pixel.

It can be considered a type of lossless compression because for certain images it can reduce the total space used while preserving the original image.

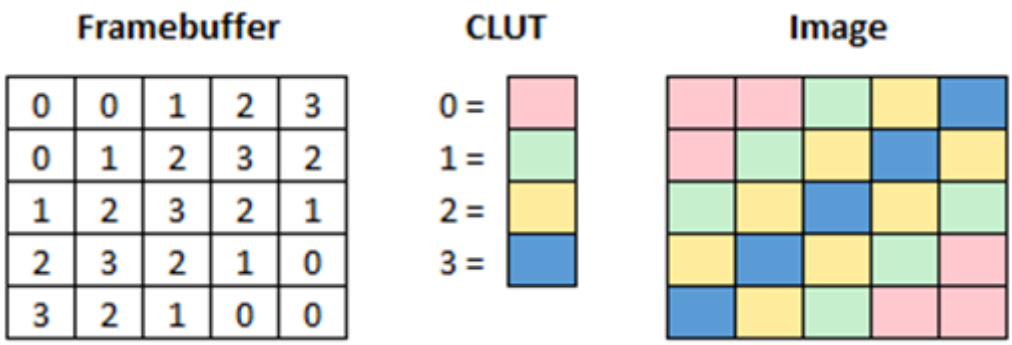


Figure 5. An image stored as a CLUT and indexed pixels

RLE

Run length encoding is a type of lossless compression because no data is lost in the process of compressing and decompressing an image. It is advantageous because it attempts to save permanent storage space (FLASH or ROM) while preserving the exact quality of the original image.

More detailed information about RLE can be found in application note AN4672: Using the Run-Length Decoding Features on Vybrid Devices.

Framerate

The framerate is the number of frames that are rendered in one second, measured in fps. Framerate may vary as the complexity of the scene increases and depends mostly on the capabilities of the devices processing the graphics (GPU, CPU, blitting engine).

Refresh Rate

The Refresh rate is the frequency at which the display is being updated. Most displays have a fixed refresh frequency by the specification and many use a range between 50-60 Hz. On systems where the framerate is different to the refresh rate double buffering may be required.

Framebuffer

A framebuffer is a memory space used to store the final image that will be shown on screen. Typically, the framebuffer size will depend on the bit depth and dimensions of the output image. For example a 480x272 24bpp will require 391680 bytes of memory.

Double buffering

Keeping two separate framebuffers allows drawing on one while the other one is shown on screen. This is typically used when separate hardware works together in a graphics application; for example a GPU composes the image and draws in one buffer while the display driver is driving the monitor using the other buffer. Separate buffers allow for race conditions to be eliminated.

Another case exists for camera sensors. Usually a camera and the display you want to show live video on operate at different speeds (framerates). Thus triple buffering is needed; The camera can fill a first buffer at its own speed while the display operates at a double buffer scheme.

Triple Buffering

Tripple buffering becomes a necessity when it is required to synchronize a video stream where framerate cannot be controlled with a double buffer rendering system. Only the video stream is required to reside in a triple buffer system, which allows having a glitchless instance of each video frame at any moment. On a triple buffered system each of the three buffers are classified in the following way:

1. View buffer: is the buffer that has been recently finished and can be used as source for the display driver
2. Drawing buffer: is the buffer which is currently being written by the video stream
3. Next buffer: is the buffer which will be used once the current drawing buffer is finished

Vertical blanking

The vertical blanking period is a time when the display is idle in between frames. This idle time is important since it is a time when there is no activity on the RGB bus (no pixels are being output to the display) and thus the processor can be used to update any of the graphics data.

Backlight

The sub pixel components that form each visible point in a screen do not emit light by themselves. Instead they form a color by defining how transparent is each of the sub pixel components, and they rely on the backlight to shine light through them so the pixels can be visible to the user. The backlight is a high power white LED array (or fluorescent tube on some screens) that is built inside the screen and it is designed to deliver white light to all of the pixels in an uniform way.

Typically, a high voltage power source is required to power the backlight of a screen. Some screens have this backlight circuit integrated, while others need an external backlight circuit. Freescale provides integrated circuits that implement backlight requirements in a flexible way. They even feature PWM inputs for dimming of the screen by modulating the brightness of the backlight.

For more information regarding Freescale's backlight driver solutions please visit Freescale / Analog and Power Management / Power Actuation / LED Drivers

Pre-rendered graphics versus dynamic graphics

A common question when developing embedded graphics systems is "can this device do 3D/reflection/particular effect?". The answer is not always straightforward, as it depends on the final look of your application. Let's say you need to display a rotation animation. If you know that your application will only display a finite number of angles of a model, and you have enough flash memory available you may be able to pre render the 3D animation and display it using a very low cost device. However, if in the application you need to display a 3D model in which any angle may be viewed then you likely need to render this scene live, which will require one of the more high end devices that include a GPU (Graphics Processing Unit).

Display controllers and interfaces

TFT displays are usually delivered in a module which includes an embedded display controller. The display controller purpose is to drive the internal signals that control how the pixel array and simplify the interface used between the microcontroller and the display. Two of the most common interfaces are described below.

- Serial IF Display Controller: This type of controllers usually have embedded video memory and allow the programmer to load images, text using a low speed interface such as SPI. Commonly used on small displays and low performance applications.
- RGB Parallel display controller: This controller requires the host device to drive the RGB data for each pixel, the pixel clock, and the control signals (HSYNC, VSYNC) for more details check the TFT Signals section. This is the most common inteface on embedded applications although is limited as the pixel clock frequency increases.

TFT signals

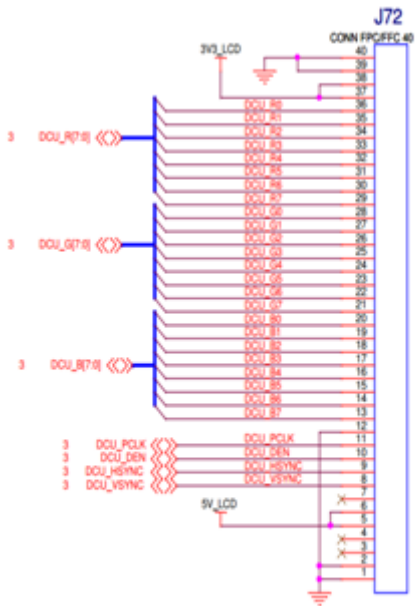


Figure 6. Typical connection to a TFT display using RGB bus

Connection to a parallel RGB interface display is usually very simple. It consists of the RGB bus (8 or 6 signals per color, depends on the screen's color depth), and the control signals which are:

- Pixel clock: indicates that new pixel data can be fetched from the RGB bus
- Data enable: indicates that the RGB bus data is valid
- Horizontal sync: indicates that the current horizontal line of pixels is finished. The next pixel transmitted corresponds to a new horizontal line.
- Vertical sync: indicates that the current frame is finished. The next pixel transmitted corresponds to a new frame.

2 Freescale graphics devices

The following sections enlist some of the Freescale devices that have at least some degree of hardware support for embedded graphics applications. Performance, price, and features vary and it is left to the reader to determine which device best fits a given application.

2.1 MPC5606S

The MPC5606S is a 32-bit power architecture microcontroller that addresses color thin film transistor (TFT) displays in automotive instrument cluster applications. It offers a cost-effective, entry level instrument cluster solution with the ability to scale designs to fit performance needs.

2.2 MPC5645S

The MPC564xS family is the next generation platform to follow the highly successful MPC560xS family designed to meet the needs of automotive instrument cluster market. The platform architecture includes on-chip display control units that directly drives the TFT displays.

Further, there is an industry standard graphics accelerator available to help drive sophisticated graphics such as simulated instrument cluster needles or image warping for heads-up displays. In addition to large built in memory, Quad SPI and DDR interfaces facilitate the possibility of accessing additional external memory. Finally, MPC564xS products offer code compatibility to customers already developing applications in the MPC560xS environment.

2.3 Vybrid

The Vybrid VF6xx family is a dual-core, that is, ARM® Cortex®-A5 and ARM Cortex-M4 based solution, with 1.5 MB on-chip SRAM, DDR2/3 and dual XiP quad SPI memory interfaces, dual high-speed USB with PHY, dual Ethernet with an L2 switch, and a digital or analog video camera interface. They are ideal for solutions that concurrently run a commercial operating system such as Linux® on the Cortex-A class core and an RTOS like MQX™ on the Cortex-M class core. By combining ARM Cortex-A5 and Cortex-M4 cores, Vybrid VF6xx devices often eliminate the need for an external MCU or field-programmable gate array (FPGA).

Vybrid devices are ideal for applications including simple HMI in appliances and industrial machines, secure control of infrastructure and manufacturing equipment, energy conversion applications such as motor drives and power inverters, ruggedized wired and wireless connectivity, and control of mobile battery-operated systems such as robots and industrial vehicles. Vybrid devices also provide a powerful combination of on-chip encryption, secure boot, anti-tamper and anti-clone capabilities to secure sensitive or critical infrastructure applications such as smart grid or industrial control.

2.4 MAC57Dxx

The MAC57Dxxx family is the next-generation platform of devices specifically targeted at driver information systems (DIS) market using single and dual high resolution displays.

Leveraging the highly successful MPC56xxS product families, our next generation product families powered by ARM processors, coupled with 2D graphics accelerators, Head Up Display (HUD) warping engines, high resolution displays, integrated stepper motor drivers with patented stepper stall detect offering leading-edge performance and scalability for cost-effective applications.

2.5 Kinetis K70

The Kinetis K7x MCU family includes 512KB-1MB of flash memory, a single precision floating point unit, Graphic LCD Controller, IEEE 1588 Ethernet, full- and high-speed USB 2.0 On-The-Go with device charge detect, hardware encryption, tamper detection capabilities and a NAND flash controller. 256-pin devices include a DRAM controller for system expansion. The Kinetis K7x family is available in 256-pin MAPBGA packages.

2.6 QorIQ LS1021A

The QorIQ LS1 family, which includes the LS1021A communications processor, is built on Layerscape architecture, the industry's first software-aware, core-agnostic networking architecture to offer unprecedented efficiency and scale.

The QorIQ LS1021A processor features an integrated LCD controller, CAN controller for implementing industrial protocols, DDR3L/4 running up to 1600 MHz, integrated security engine and QUICC Engine, and ECC protection on both L1 and L2 caches. The LS1021A processor is pin- and software-compatible with the QorIQ LS1020A and LS1022A processors.

2.7 i.MX family

The i.MX family of processors represents Freescale's next generation of advanced multimedia and power-efficient implementation of ARM Cortex cores with core processing speeds up to 1.2 GHz. All of the i.MX devices are optimized for both performance and power to meet the demands of high-end, advanced applications. Ideal for a broad range of applications in the consumer, automotive, medical, and industrial markets, the i.MX includes an integrated display controller and enhanced graphics and connectivity features

2.8 Quick reference comparison

The following tables compare some of the similar devices so that the reader can quickly check the features for choosing a platform considering application requirements. Please note that not all Freescale graphics devices are included in this comparison.

Table 2. Graphics capabilities comparison

	MPC5606S	MPC5645S	VYBRID	MAC57Dxx
Flash memory	1 MB + external QSPI	2 MB + external QSPI	0	4 MB + external QSPI, hyperflash
RAM	0.2 MB	1.06 MB + external DDR2	1.5 MB + external DDR3	3.3 MB + external DDR2
Composition unit	1 x DCU	2 x DCU	2 x 2DACE	2 x 2DACE + HUD + Writeback
CPU	PowerPC e200z0h @64 MHz	PowerPC e200z0h @124 MHz	ARM CortexA5 500 MHz CortexM4167 MHz	ARM CortexA5 320 MHz CortexM4160 MHz + CortexM0
GPU	n/a	OpenVG GPU	OpenVG GPU + TinyUI	OpenVG GPU + TinyUI
Video input	PDI	VIU2	VIU3	VIU4

Table 3. Graphics features comparison

	MPC5606S	MPC5645S	VYBRID	MAC57Dxx
Display controller	16 layers 4 plane blend	16 layers + 4 layers 4 plane blend	64 + 64 layers 6 plane blend	32 + 32 layers 6 plane blend

Table continues on the next page...

Table 3. Graphics features comparison (continued)

	MPC5606S	MPC5645S	VYBRID	MAC57Dxx
2D Graphics	Prerendered	Dynamic	Dynamic	Dynamic
Target resolution	1x VGA (640x480)	2x XGA (1024x768)	2x WXGA (1366x768)	2x WXGA (1366x768)
Hardware rendering	n/a	OpenVG GPU	OpenVG GPU	OpenVG GPU + Writeback
HUD	n/a	n/a	n/a	HW HUD warping

3 Graphics animation using the 2D-ACE

The 2D-ACE (Two Dimensional Animation and Composition Engine) is a system master that fetches graphics stored in internal or external memory and displays them on a TFT LCD panel. A wide range of panel sizes is supported and the timing of the interface signals is highly configurable. Graphics are read directly from memory and then blended in real-time, which allows for dynamic content creation with minimal CPU intervention. Graphics may be encoded in a variety of formats to optimize memory usage.

This section intends to explain some animation concepts so that the reader can implement animation code based on the DCU/2D-ACE modules. This section is not intended to be an in-depth look at the DCU or the 2D-ACE. For your reference the following application notes are recommended for configuring the DCU/2D-ACE:

- AN4444 : Configuring and using the DCU3 and DCULite on the MPC5645S MCU.
- AN4865 : TFT Panel support in the MPC5645S Microcontroller Family
- AN4187 : Configuring and using the DCU2 on the MPC5606S MCU
- AN4647 : Configuring and using the 2D-ACE on Vybrid Microcontrollers

Animations and interpolations

Since the 2D-ACE provides a very simple interface for manipulating key image properties (position, transparency, foreground and background colors) animating a graphics is a very simple operation, it just consists in writing to the appropriate register.

For example, modifying the 32 bit register that controls the position to a value of 100 from an original value of 50 will cause the image being displayed on layer 1 to move 50 pixels to the right in the next frame. Similarly any other property could be modified such as the transparency.

Obviously such an abrupt change is possible, but it is likely undesired as current requirements of user interfaces call for smooth, high quality transitions and animations. To achieve we need to modify the properties of the layers (graphic objects) in a better way, we need to interpolate over a number of frames.

Interpolation is a method of constructing new data points within the range of a discrete set of known data points; our known data points are the current value and the desired value. We know where the image is, where we want it to be, and how long we want it to take to get there. The interpolation will set the tone of the animation: it is up to the interpolation to see that the object, accelerates, brakes, both, or even if it moves in an elastic or bouncy manner.

By interpolating we will provide a new position or transparency each frame for the graphic that we want to animate. Given the framerate of our system we can calculate how much frames we want to take so that the animation takes a desired amount of time. Example: most graphics systems have a 60 Hz framerate so an animation that takes 60 frames to finish will take 1 second.

Position animation

A position animation consists in smoothly moving a graphics from a starting position to a final position (x_1,y_1) . There are several different processes to achieve this that are called interpolations. The different types of interpolations will produce different visual effects.

The simplest type of interpolation is the linear interpolation. It consists of simply adding a fixed delta to the x and y properties of the graphic until it reaches the final position.

$$\Delta x = (x_1 - x_0) / N_{frames}$$

Equation 2.

$$\Delta y = (y_1 - y_0) / N_{frames}$$

Equation 3.

$$x_{new} = x_{current} + \Delta x$$

Equation 4.

$$y_{new} = y_{current} + \Delta y$$

Equation 5.

This produces an animation that has a constant speed. This kind of animation is very simple. Other interpolations can produce more stunning effects such as easing.

$$x_{new} = x_{current} + (x_{final} - x_{current}) * speed$$

Equation 6.

Acceleration and braking is easily implementable

$$x_{new} = x_{current} + \Delta x$$

Equation 7.

$$\Delta x_{current} = \Delta x_{previous} + acceleration$$

Equation 8.

Fading animation

Fading can be implemented by modifying the ALPHA property of an image. An initial alpha of 0% and gradually incrementing this ALPHA until it reaches 100% or another desired final value will produce a fade in. A fade out is similarly implemented by starting with an ALPHA value of 100% or other non zero value and gradually decrementing it each frame until it reaches 0%.

ALPHA values can also be interpolated, linearly or otherwise as seen in the previous section, to achieve more stunning effects instead of just modifying the ALPHA at a constant rate, which would be a linear interpolation.

Color morphing animation

Since the 2D-ACE generates the colors used by transparent images by means of the foreground and background colors, we can modify each frame so that the visual effect to the user is that of a smoothly changing color.

Similarly to position and transparency animation, foreground and background colors can also be interpolated, linearly or otherwise.

More importantly, any of these types of animations can be combined to achieve more complex effects; for example an image could appear on screen by entering the visible area moving from left to right, while at the same time fading in (moving from transparent to opaque) to achieve a “fly in” effect.

Flipbook animation

A flipbook animation consists in changing the displayed image each frame to produce a frame by frame or flipbook effect. It can be seen as a “pre rendered” animation, or “video”. A flipbook animation may take up a lot of permanent storage memory (flash) in the final application, but it provides the advantages of very low CPU/GPU overhead, and it can be used to provide pseudo 3D graphics, or rotation and other effects that are not possible to render on the fly on low-end platforms.

4 OpenVG and OpenGL

OpenVG and OpenGL are open standards for acceleration of graphics processing. They are available on a wide variety of platforms and devices. This section intends to give a brief introduction on both API's, which are available for use on select Freescale devices. OpenVG is intended to accelerate 2D and vector applications, while OpenGL is intended for 3D applications. For more details on the standards refer to www.khronos.org.

OpenVG

The OpenVG engine can draw and translate images using vector data or source raster images and in both cases, significant memory savings can be made by manipulating the images as they are needed for the display.

OpenVG is a standard created by the Khronos Group (who also created OpenGL and OpenGL ES), for the need of a powerful low-level 2D vector graphics API. There are many applications that can take advantage of accelerated 2D graphics (for example, portable mapping and GPS applications, portable media players, high end 2D accelerated games, advanced user interfaces, and screen savers).

The OpenVG vector-based accelerated standard are used to display many formats that are in the nature of vector graphics, such as Flash, SVG, PDF, PostScript, and vector fonts. This standard makes use of several functions that are low-level, similar to the OpenGL functions, but based on Bezier curves, rather than polygons.

Freescale provides the following documentation for diving more deeply into the OpenVG topic:-

- An Introduction to OpenVG
- Using OpenVG for 2D graphics in auto infotainment displays

OpenGL

OpenGL is the most widely adopted 2D and 3D graphics API in the industry, bringing thousands of applications to a wide variety of computer platforms. It is window-system and operating-system independent as well as network-transparent. OpenGL enables developers of software for PC, workstation, and supercomputing hardware to create high-performance, visually compelling graphics software applications, in markets such as CAD, content creation, energy, entertainment, game development, manufacturing, medical, and virtual reality. OpenGL exposes all the features of the latest graphics hardware.

The OpenGL GPU (GPU3D) in the i.MX devices is a high performance core that delivers hardware acceleration for 3D graphics applications. Addressable screen sizes range from the smallest cell phones to HD 1080p displays. It provides high performance, high quality graphics, low power consumption, and the smallest silicon footprint. GPU3D accelerates numerous 3D graphics applications, including graphical user interfaces, menu displays, flash animation, and gaming. The GPU3D in the i.MX processors support the following API's:

- OpenGL ES 1.1, 2.0, 2.1, and 3.0
- OpenVG 1.1
- EGL 1.4
- DirectX 11.9.3
- OpenCL 1.1 E

The following Freescale application note is available for getting started with OpenGL and using Freescale devices:

- AN4274: High end 3D graphics with OpenGL ES 2.0

5 References

AN4672: *Using the Run-Length Decoding Features on Vybrid Devices*, available in freescale.com.

AN4444: *Configuring and using the DCU3 and DCULite on the MPC5645S MCU*, available in freescale.com.

AN4865: *TFT Panel support in the MPC5645S Microcontroller Family*, available in freescale.com.

AN4187: *Configuring and using the DCU2 on the MPC5606S MCU*, available in freescale.com.

AN4274: *High end 3D graphics with OpenGL ES 2.0*, available in freescale.com.

Compositing Digital Images Computer Graphics Porter, T. and T. Duff. Vol. 18, Number 3, July 1984. pp 253-259

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN5072
Revision 0, 02/2015