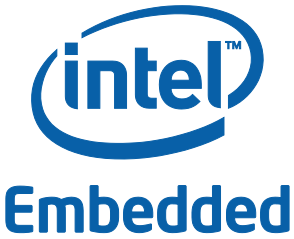


White Paper
Nuhairi Anuar
Software Engineer
Intel Corporation
Ho Nee Shen
Senior Software Engineer
Intel Corporation

Framebuffer Overlay Blending

Configuration and Proof of Concept with Intel® EMGD

November, 2010

324707-001

Executive Summary

Intel® Embedded Media and Graphics Driver (Intel® EMGD) provides an embedded, focused graphic solution for Intel low-power integrated graphics. Intel® EMGD supports a variety of features for graphic applications, such as video overlay and accelerated 3D graphics for various commercial operating systems such as Linux*, MeeGo*, and Windows* XP.

Using a feature called Framebuffer Overlay Blending, the Intel® EMGD design allows the driver to be as flexible as possible without sacrificing performance. This feature provides visually appealing blending of video overlay with 3D graphics using one of the supported 3D APIs such as OpenGL, OpenGL ES, or Direct3D*.

This paper describes the methods for enabling the Framebuffer Overlay Blending feature in Intel® EMGD and shows a proof of concept for the feature. This paper assumes that the user is familiar with basic Intel® EMGD configuration using the Intel® EMGD Configuration Editor (CED). It explains each of the configurations available in CED that are related to Framebuffer Overlay Blending feature.

This paper shows a proof of concept on how this feature will be used by showing an OpenGL application blended over a video overlay. It also includes an example on how to modify simple 3D applications to be used with Framebuffer Overlay Blending. It provides common troubleshooting tips and techniques when enabling and using Framebuffer Overlay Blending.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. www.intel.com/embedded/edc.

Contents

Background

Display Engine Components

The main function of the display engine is to composite the various post-processed components in the graphics adapter and stream the data to the display device, such as an LCD monitor. The display engine consists of planes, pipes, and ports. In the Intel® Atom™ processor E6xx there are seven display planes, two display pipes, and two display ports, LVDS and SDVO.

Display planes are regions in the graphics memory that act as render targets in the display adapter. The planes hold the post-processed image that is ready to be displayed by the monitor. The planes also store the post-processed image in the proper pixel format. Examples of post-rendered images are mouse cursor, video frames after the raw data has been decoded, and a 3D image after it has been rendered by the 3D engine.

The planes are

- Display Plane A
- Display Plane B
- Cursor A
- Cursor B
- VGA plane
- Video Overlay
- Sprite C/Display Plane C

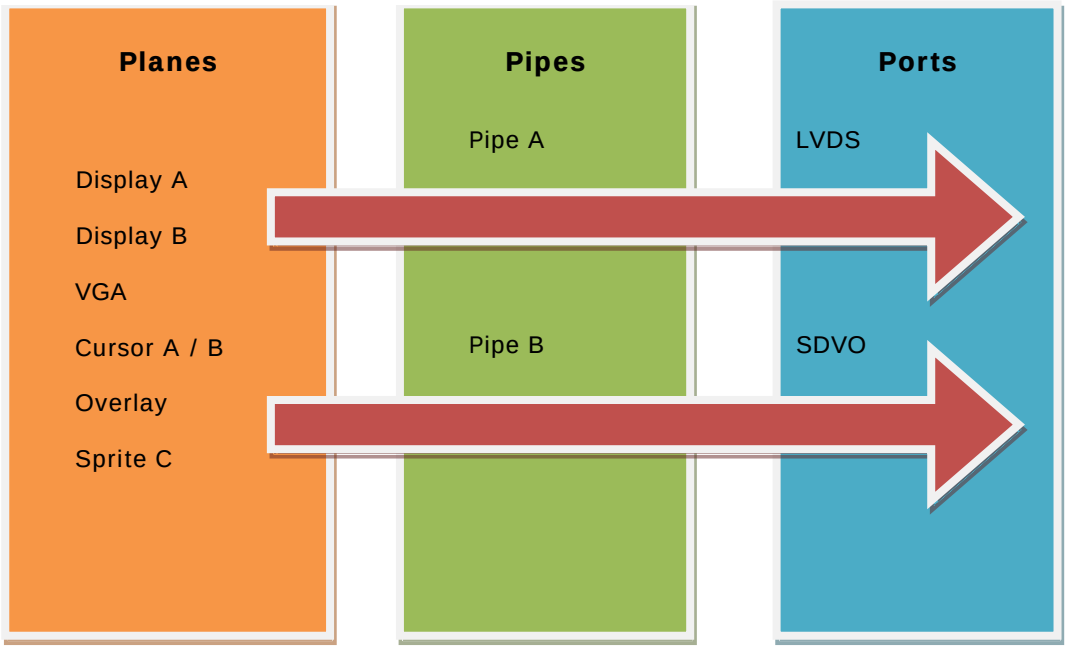
The display pipes take the various display planes and composite them into a single data stream to be displayed. The process of compositing the display planes is also known as *blending*. The pipes also handle the timing, adding the blanks and syncs into the data stream.

The display ports control the display device specific interface, such as LVDS and SDVO. The port converts the signals coming from the pipe into the display device-specific electrical signals. For example, LVDS ports convert the signals from the display pipe into LVDS interface signals.

The Framebuffer Overlay Blending feature uses the functionality of planes and pipes to create a semi-transparent effect on top of video.

Note: In the Intel® Atom™ processor E6xx, Pipe A is explicitly connected to the LVDS port and Pipe B is connected to the SDVO port.

Figure 1: Data flow of the display engine and components



Framebuffer

The framebuffer or display plane is a location in graphic memory that stores the whole image that should be shown on a physical display. All 2D and 3D operations finally write to this section of memory so that it can be displayed on the screen. The framebuffer is analogous to the desktop on the computer.

Video Overlay

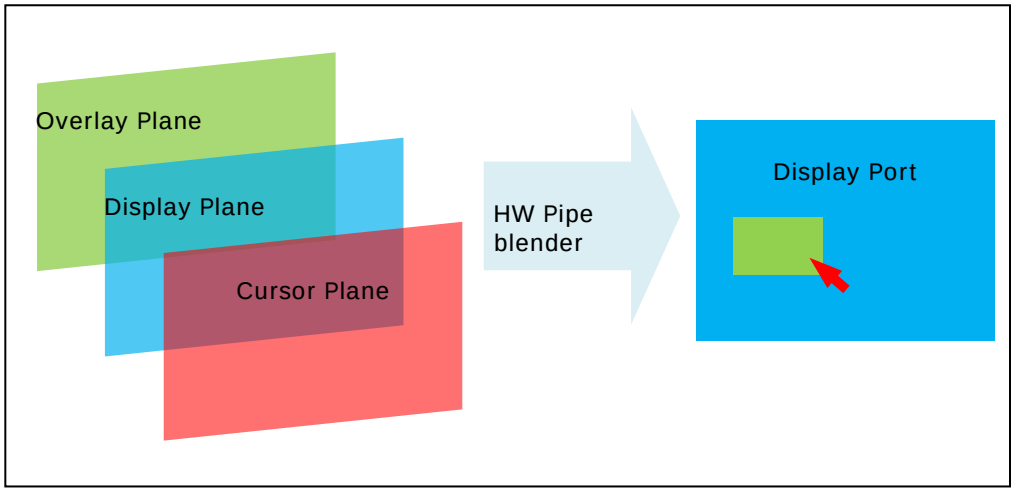
Intel® EMGD offers a video overlay feature to speed up video display. The overlay acts as the render target for video playback and holds the decoded video frame. From the user perspective, the video overlay appears to be on top of the graphic framebuffer (display plane). However, that overlay plane is located last in Z-order where it is virtually behind the framebuffer plane and the cursor plane. In the traditional method, the framebuffer overrides all overlay pixels in a pipe except where the “color key” on the framebuffer matches a specific color.

The Intel® Atom™ processor E6xx and Intel® System Controller Hub US15W chipset each have only one video overlay plane. Therefore, only the first video playback application to be executed, such as Microsoft* Media Player* or CyberLink* PowerDVD*, will use the overlay. Any additional video playback will have its contents drawn directly into the framebuffer (display plane). When the first video playback application that was using the overlay closes, the next video playback application can then use the overlay plane.

Checking Video Playback on the Overlay Plane

The easiest way to check whether video playback is using the Overlay Plane is to see the results of a screen capture. When running the video, capture a screen (in Windows XP, press "print screen" on the keyboard) and paste the screen capture into a picture editor, for example, Paint in Windows XP. The area where the video playback is displayed should show the color key (a box with a single color) when the Overlay Plane is being used. You should not see the video frame if the playback is done on the Overlay Plane.

Figure 2. Overlay, Display and Cursor Planes



3D Graphics

Intel® EMGD running on the Atom E6xx and US15W offers advanced 3D graphic acceleration. This enables customers to use various APIs such as OpenGL, OpenGL ES, Direct3D, etc. Regardless of API, 3D graphics will be rendered to the framebuffer (display plane).

Framebuffer Overlay Blending

With the Framebuffer Overlay Blending feature, Intel® EMGD offers an opportunity for customers to exploit Intel's hardware capabilities. Framebuffer blending of 3D graphics with overlay does not cause any significant increase in CPU or GPU usage compare to other methods such as compositing. One example of this usage model is having control buttons (play, rewind, and pause) displayed semi-transparently over the video playback.

Alternatively, to achieve the same outcome as Framebuffer Overlay blending, a customer could write an application that blends the video and 3D into the framebuffer. This means that the application would need to play the video into the framebuffer and then have the 3D application drawing onto the video. With this alternative method the video could not be played on the overlay plane, which reduces video playback performance. Secondly, there is additional processing to blend the 3D render target on top of the video playback.

Requirements

Framebuffer Overlay Blending is currently supported on Intel® Atom™ processor E6xx and Intel® System Controller Hub US15W chipset. The system needs to run Intel EMGD and be configured for Framebuffer Overlay Blending (refer to Intel® EMGD Configuration on page 9).

The media application needs to play the video using the overlay plane. Most applications, by default, play video in overlay. Please refer to the video player manual if unsure.

This feature should work on all 3D API and video formats as long as the video playback is done on the Overlay. This feature is not available on Sprite C. From the Intel® EMGD implementation perspective, Framebuffer Overlay Blending is not available in the secondary display of Clone mode.

Intel® EMGD Configuration

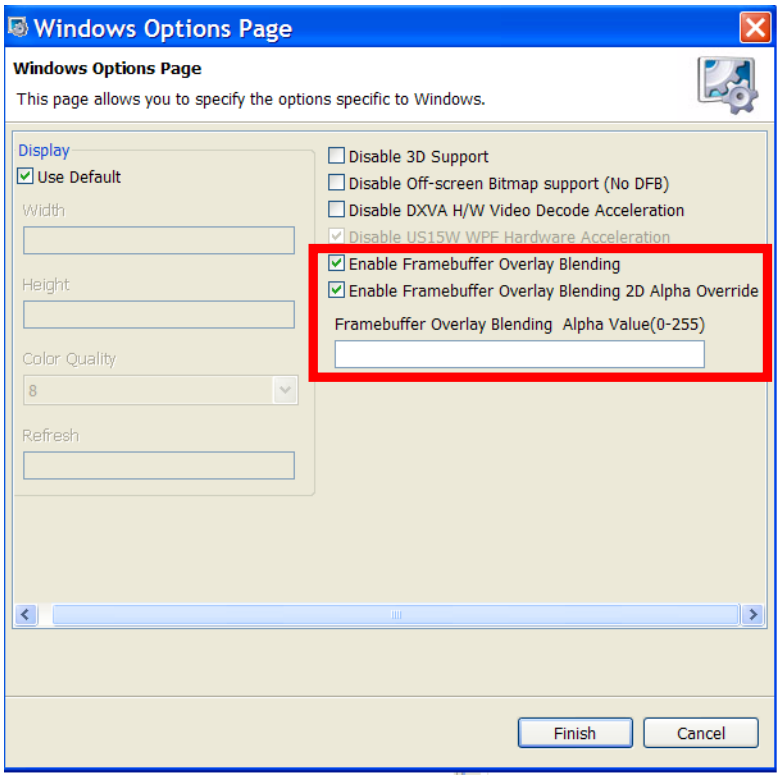
The user needs to configure Intel® EMGD to enable the Framebuffer Overlay Blending feature, as described below.

Windows XP

There are three configurations needed for Windows XP:

- Enable Framebuffer Overlay Blending – This option enables the hardware to blend the framebuffer with the overlay based on the alpha value of each framebuffer pixel. The customer sees the desktop in Windows XP turn black when turning on overlay if the following two options are not enabled. This is because most of Windows XP 2D operation does not carry alpha value.
- Enable Framebuffer Overlay Blending 2D Alpha Override – This will enable a workaround in the driver to override the value of all 2D pixels alpha value to the value specified in following option.
- Framebuffer Overlay Blending Alpha Value (0-255) – This value will be OR'd with all 2D pixels alpha value before written to the framebuffer. This will eliminate the black "corruption" in Windows XP. Most of the time the user will set an alpha value of 255. With a value of less than 255, any 2D surface will be transparent when placed on top of video overlay.

Figure 3. CED – Package Option for Windows XP



CED outputs the `iegd.inf` file containing these lines:

```
HKR, ALL\7\General , FbBlendOvl, %REG_DWORD%, 1
HKR, ALL\7\General , FbBlend2DAlphaOverride, %REG_DWORD%, 1
HKR, ALL\7\General , FbBlendAlphaValue, %REG_DWORD%, 0xFF
```

2D Alpha Override Feature

Intel designed a workaround that overrides all 2D pixels' alpha values for customers who use this feature in Windows XP. This feature enables Windows XP customers to use the Framebuffer Overlay Blending feature without seeing any corruption due to a missing alpha value in most of Windows XP 2D operations. However, the user should expect a minor increase in CPU usage due to the overriding operation. This feature has several other limitations:

- Windows XP display must be set to 32 bits.
- Microsoft WHQL may fail if it is tested when this feature is turned on.
- This feature will slow down 2D operation slightly. Most of the time it is not noticeable to the user.
- If the overriding value for the alpha is set by the user to a value other than 255, the user will see any 2D surface that is drawn on top of overlay video as semi-transparent depending on the value set.

Figure 4. 2D appears semi-transparent on top of video overlay when overriding value is set to 128 (50% transparent)

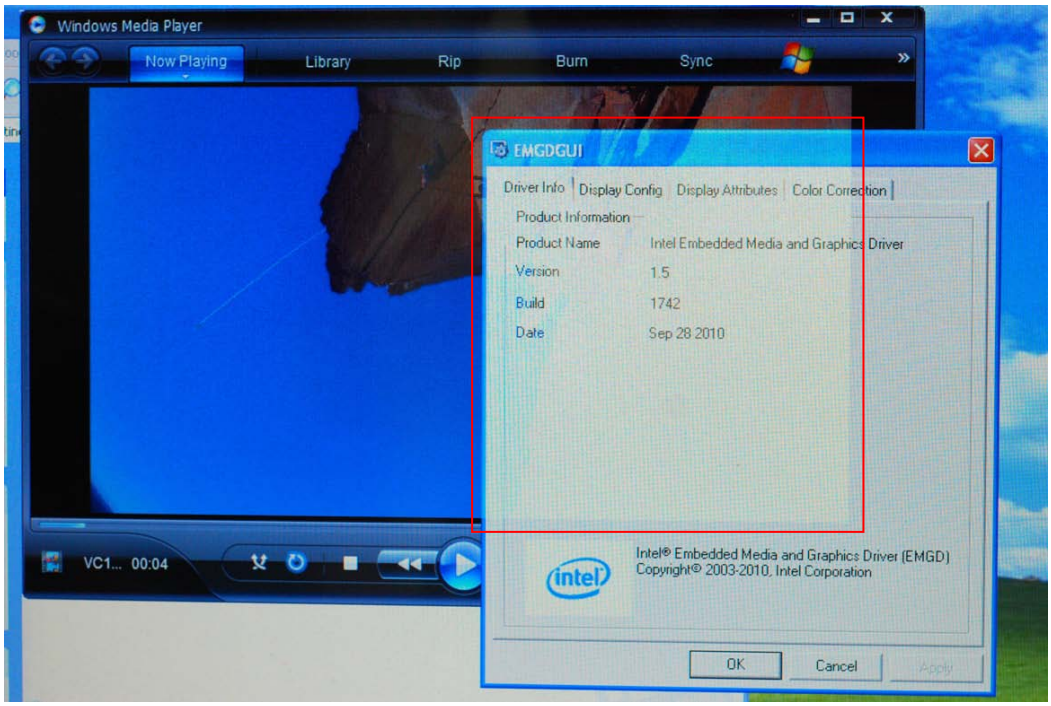
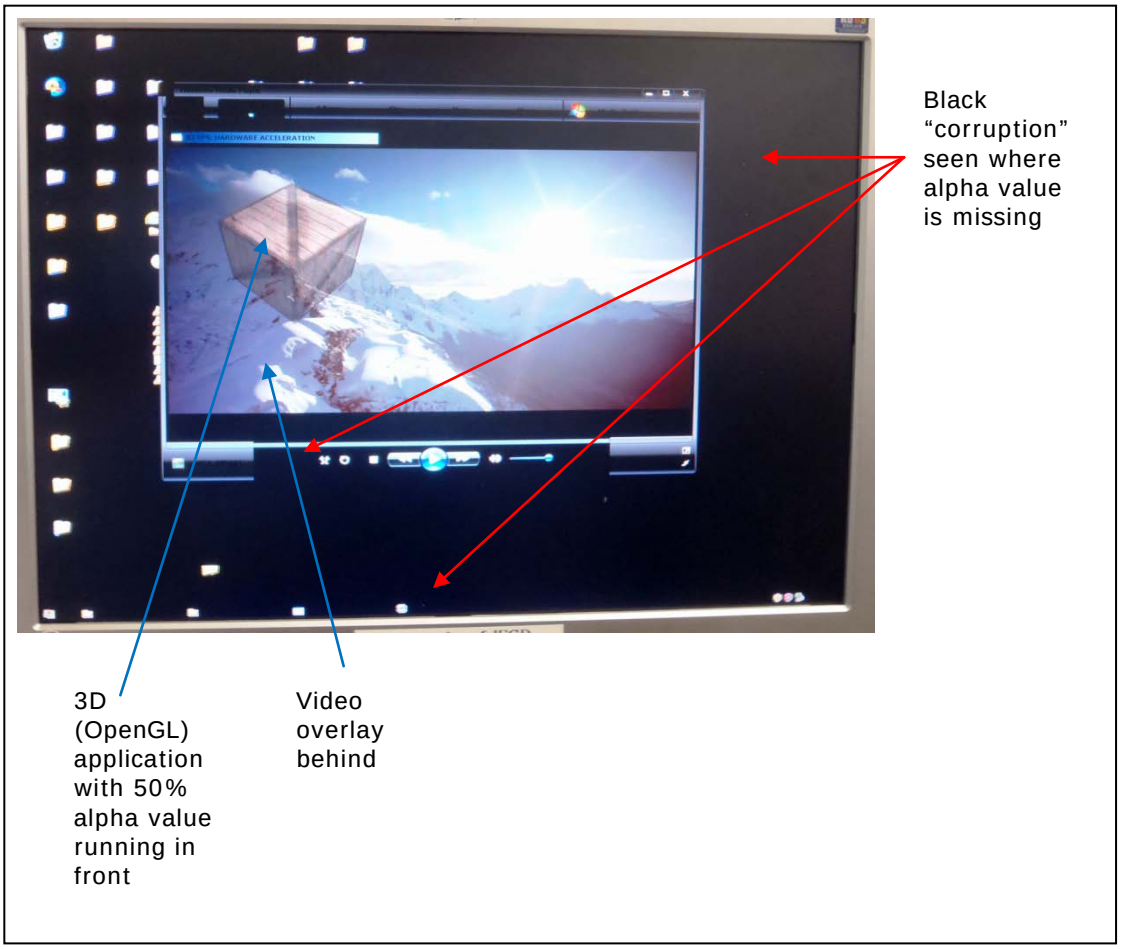


Figure 5. Black “corruption” with 2D Alpha Override feature OFF and Framebuffer Blend Overlay ON

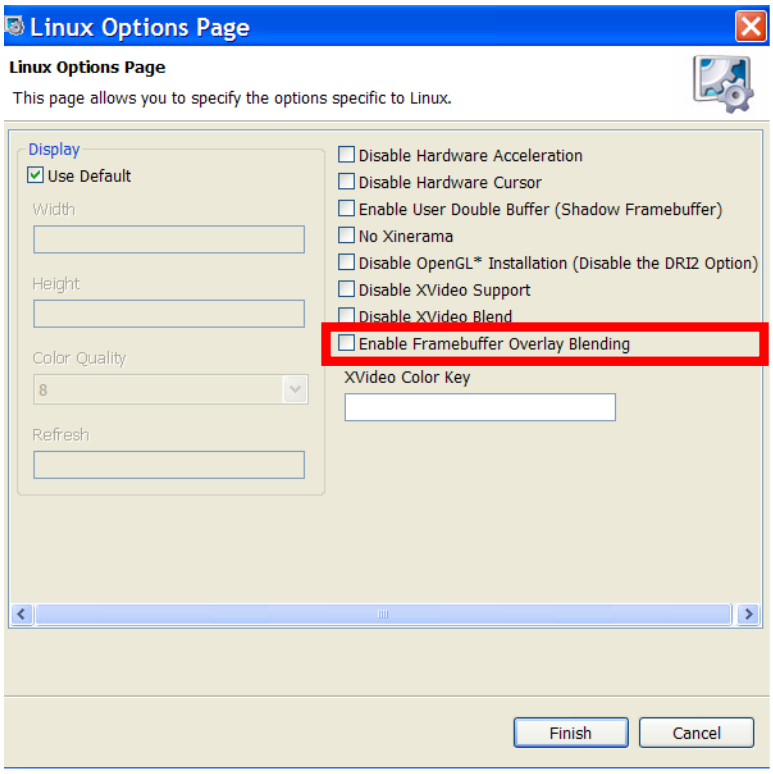


Linux and MeeGo

There is only one configuration for MeeGo and all supported Linux distros:

- Enable Framebuffer Overlay Blending – This enables the hardware to blend the framebuffer with overlay based on the alpha value of each framebuffer pixel.

Figure 6. CED – Package Option for Linux



Note: Intel® EMGD currently does not support 2D alpha value overriding feature on any Linux OS or MeeGo.

CED outputs `xorg.conf` containing this line:

```
Option    "ALL/7/General/FbBlendOvl"    "1"
```

Framebuffer Overlay Blending Formula

What is seen by the user as the product of the blending process is calculated using the following formula for each pixel on the display.

OutP = ((DisplayPlane.Alpha / 255) * DisplayPlane.RGB) +

(((255 – DisplayPlane.Alpha)/255) * OverlayPlane.RGB)

OutP = Visible output pixel on display at location X,Y

DisplayPlane = Display plane (Framebuffer) pixel at location X,Y

OverlayPlane = Overlay plane pixel at location X,Y (after hardware conversion to RGB)

Proof of Concept

3D on Top of Video

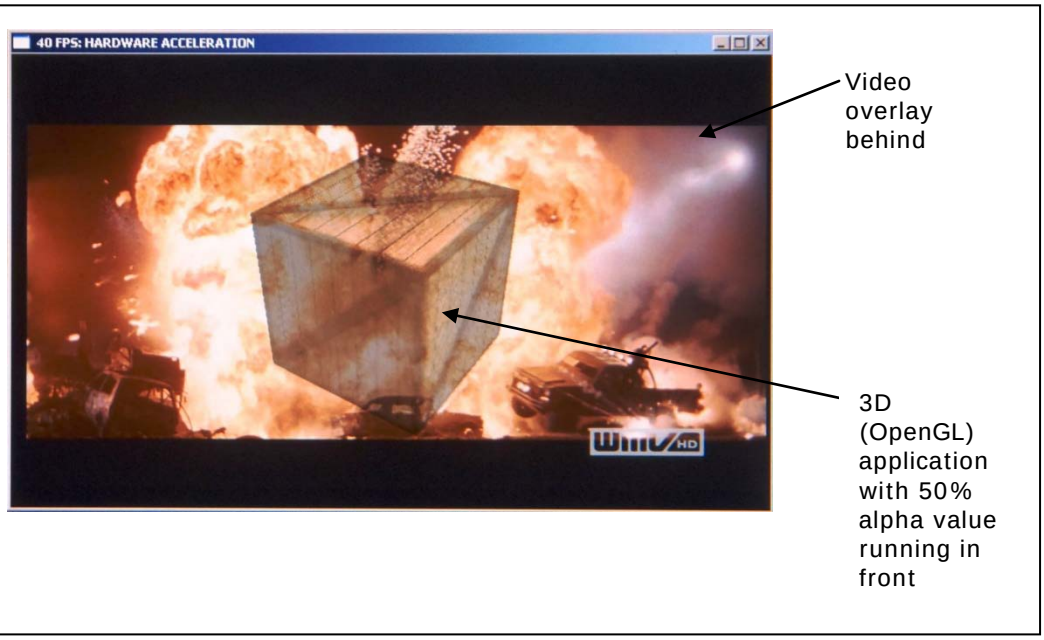
This is a demonstration on how to use the Framebuffer Overlay Blending feature. There are two components needed:

- A supported video player, one that plays video in the Overlay Plane. The overlay feature must be turned on in EMGD CED.
- A 3D application, either OpenGL or Direct3D. The object's color must contain an alpha value to have a transparency effect.

It is does not matter whether the two components are separate applications or a single application that contains both video and 3D.

Note: A user needs to modify only the 3D application (adding alpha values) to achieve this effect. No changes are necessary on the video file or video playback application.

Figure 7. OpenGL application running transparently in front of the video overlay



Modifying an Existing Simple OpenGL Application

The OpenGL application needs to have an alpha value for its colors if the user wants to use it with the Framebuffer Overlay Blending feature. Without the correct alpha value, the OpenGL application will fully overwrite the video overlay. As an example, here is a snapshot of OpenGL codes for drawing a triangle using RGBA. The function names and arguments highlighted in red are necessary for the OpenGL application to render with an alpha value. The arguments added (highlighted in red) are the alpha values in the pixel format. These arguments allow the application to be semi-transparent when it is rendered on top of the video playback.

The key to modifying an OpenGL application for Framebuffer Overlay Blending is to add semi-transparent alpha values to the vertices and background of the application. This means using `glColor4f` instead of `glColor3f` and using `glClearColor` with four parameters (last parameter being the alpha value).

In the example below, the only parts that need to be modified are the ones involving the colors of the object and the clear color option. The user can set any alpha value depending on the desired effect. The Framebuffer Overlay Blending feature will use these alpha values to calculate the output of the color using the formula presented in the previous section.

```
/*Open GL Triangle*/
#include <GL/gl.h>
#include <GL/glx.h>

static void redraw( Display *dpy, Window w ){
   glClear( GL_COLOR_BUFFER_BIT );
   /* triangle */
   glBegin(GL_TRIANGLES);
   glColor4f( 1.0, 0.0, 0.0, 0.4 );
   glVertex2f( 0, 0.8 );
   glColor4f( 0.0, 1.0, 0.0, 0.5 );
   glVertex2f( -0.8, -0.7 );
   glColor4f( 0.0, 0.0, 1.0, 0.7 );
   glVertex2f( 0.8, -0.7 );
   glEnd();
   glXSwapBuffers( dpy, w );
}

int main( int argc, char *argv[] ){
   …
   …
   …
   glClearColor( 0.5, 0.5, 1.0, 0.3 );
   …
   …
   …
   redraw(…);
   return 0;
}
```

Modifying an Existing Simple Direct 3D Application

Direct 3D is part of Microsoft's DirectX* API that allows users to write 3D applications for Microsoft's family of products. The advantage of using DirectX is the ability to easily integrate all the DirectX components with the Windows API such as windowing.

Using a simple example of a textured object in D3D, the highlighted code below shows the necessary change to add alpha values for 3D rendering.

```
VOID Render()
{
    // Clear the backbuffer and the zbuffer
    g_pd3dDevice->Clear(
        0, NULL, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,
         D3DCOLOR_ARGB(100,0,0,255), 1.0f, 0 );
    ……
    ……
}
```

Note: The Render function is called to render every frame of the 3D application. The highlighted code above clears the background of the 3D application with an alpha value. In this example, the background is painted blue with opacity of about 39% (Alpha value of 0 means full transparency). This allows the application to be semi-transparent when rendered on top of the video playback.

A user can modify the alpha value (highlighted in RED above) to control how much transparency the 3D application will experience when placed on top of the video.

```
// A structure for our custom vertex type.
struct CUSTOMVERTEX
{
    D3DXVECTOR3 position; // The position
    D3DCOLOR    color;    // The color
};

HRESULT InitGeometry()
{
    // Use D3DX to create a texture from a file based image
    if( FAILED( D3DXCreateTextureFromFile(
                g_pd3dDevice, "Texture.bmp", &g_pTexture ) ) )
    {
            return E_FAIL;
    }

    // Create the vertex buffer.
    if( FAILED( g_pd3dDevice->CreateVertexBuffer(
                3 * sizeof(CUSTOMVERTEX),
        0, D3DFVF_CUSTOMVERTEX,
        D3DPOOL_DEFAULT, &g_pVB, NULL ) ) )
    {
        return E_FAIL;
    }

    // Fill the vertex buffer
    CUSTOMVERTEX* pVertices;
    if( FAILED( g_pVB->Lock( 0, 0, (void**)&pVertices, 0 ) ) )
        return E_FAIL;

    {
        pVertices[0].position =
                D3DXVECTOR3( 150.0f, 50.0f, 1.0f );
        pVertices[0].color    = 0xc0ffffff;

        pVertices[1].position =
                D3DXVECTOR3( 250.0f, 250.0f, 1.0f );
        pVertices[1].color    = 0xc0808080;

        pVertices[2].position =
                D3DXVECTOR3( 50.0f, 250.0f, 1.0f );
        pVertices[2].color    = 0xc0ffffff;

    }

    g_pVB->Unlock();

    return S_OK;
}
```

Note: The InitGeometry function is called during the initialization of the 3D application to create the objects in the 3D application. This function also creates and maps the texture onto the 3D object. The red highlighted text above shows where changes have to be made to create a semi-transparent alpha value in the vertex color. The pixel format of the texture is not taken into account when rendering the alpha values of the object.

Troubleshooting

Refer to the table below if you encounter a problem enabling this feature.

Table 1. Troubleshooting methods

Problem	Solutions
Windows XP desktop becomes dark when playing video.	Turn on Framebuffer Blending 2D Alpha Override and set Framebuffer Overlay Blending Alpha Value to a non-zero value.
3D (OpenGL or Direct3D) is not transparent on top of video.	Check whether the overlay feature is turned on in CED. Intel® EMGD does not support overlay when the screen is set to any rotation to flip. So framebuffer blend overlay will not work when the screen is set to rotated or flipped. Make sure 3D support is enabled in CED Uninstall and reinstall Intel® EMGD. Make sure installation is successful and all binaries are in place. Make sure 3D application has correct alpha value. Make sure the video player used is supported by Intel® EMGD.
3D and video become too slow (frame skip)	There is a limitation on how much graphic processing can be done by the hardware. Limit the number of polygons and textures in your 3D application to reduce 3D graphic processing. Large video resolution is also known to cause slowdown.

Conclusion

The Framebuffer Overlay Blending feature provides an easy way for users to achieve an eye-catching display by merely enabling the feature in the graphic driver and making simple modifications to the 3D application. This feature in the Intel® Atom™ processor E6xx and Intel® System Controller Hub US15W chipset reduces the amount of CPU cycles to achieve video and the 3D blending effect compared to programming on a standard operating system API. The Framebuffer Overlay Blending feature is available for Windows XP, MeeGo, and various supported Linux operating systems. Framebuffer Overlay Blending is designed to work with various 3D graphic API such as OpenGL, OpenGL ES, and Direct3D.

Customers should be able to modify existing D3D or OpenGL applications to make use of the Framebuffer Overlay Blending feature in Intel® EMGD.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. http://intel.com/embedded/edc.

Authors

Nuhairi Anuar is a Software Engineer with Intel Embedded and Communication Group

Ho Nee Shen is a Senior Software Engineer with Intel Embedded and Communication Group

Acronyms

API – Application Programming Interface

CED – Configuration Editor

D3D – Direct3D*

EMGD – Intel® Embedded Media and Graphics Driver

LCD – Liquid Crystal Display

LVDS – Low Voltage Differential Signal

OGL – OpenGL

OGLES – OpenGL ES

OS – Operating System

SDVO – Serial Digital Video Out

§