Pluggable Window System

Implementation Guide (SGX)

Filename	:	Pluggable Window System.Implementation Guide.doc
Version	:	1.0.622 External Issue
Issue Date	:	15 Jul 2010
Author	:	PowerVR

Contents

1. Introduction

1.1. Scope

This document provides implementation guidelines for a loadable Window System EGL (WSEGL) module that provides the necessary interface between the PowerVR OpenGLES driver and the native window system. These guidelines include usage of the PVR2D functions that must be used if the customer is controlling the allocation of the drawable buffer memory pages.

The module will be responsible for such things as creating window drawables, enumerating display capabilities and providing any surface information required by OpenGLES to perform a render.

1.2. Intended Audience

The audience of this document should be familiar with the following APIs and systems:

- Khronos EGL
- OpenGLES
- Windows System such as Series 60, the X11 Window system, JavaVM, or WinCE Window System

1.3. Related Documents

PowerVR 2D.PVR2D API.doc

Khronos OpenGLES specification. http://www.khronos.org/opengles/spec.html

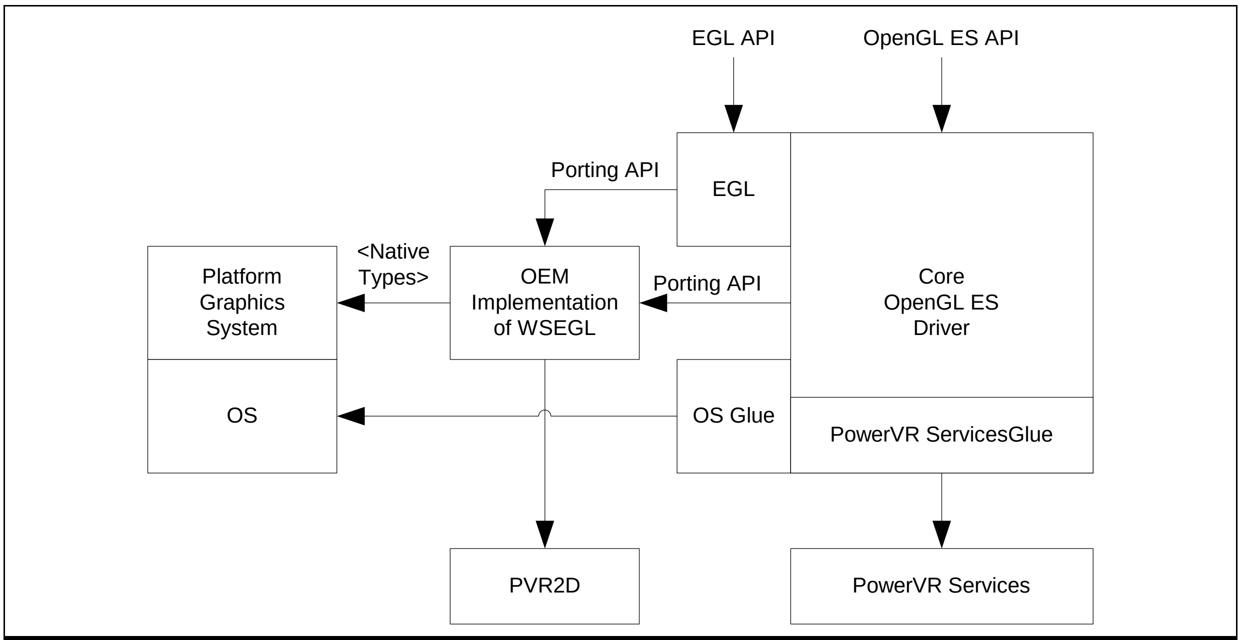
Khronos EGL specification. http://www.khronos.org/opengles/spec.html

1.4. Terminology

Term	Meaning
OpenGL®-ES^{TM}	The 3D drawing API that is defined by the Khronos Group, and which is being facilitated through this document. OpenGL is a registered trademark of Silicon Graphics Inc.
Application	The software that calls the egl*() and gl*() entry points
WSEGL library	A module that implements the APIs described in this document
OEM implementation	An OEMs particular implementation of the WSEGL library

2. System Description

2.1. Architecture/System diagram



OGL/EGL/WS/Services Interaction

2.2. Introduction

The WSEGL is a set of Porting API's that the OEM implements to provide an interface between the OpenGL ES Driver and the platform Native Types. The Native Types may be specific to the platform and not defined by an external body.

2.3. Native Types

Most window systems provide for the concept of a Display, a Window and an off-screen image or Pixmap. The Khronos EGL spec was designed to allow these objects to be typedef'd as the NativeTypes required for the EGL API. To implement this API the OEM implementation must create appropriate typedefs for the Khronos EGL Native Types, these are:

- NativeDisplayType
- NativePixmapType
- NativeWindowType

The types that are defined must fit into the storage provided by a 'C' void * as these are passed through the EGL API in the OpenGL ES driver. That means that in general these types must themselves be pointers, or integers less than or equal to the storage size of a pointer.

The NativeTypes are transparent to the OpenGL-ES driver, and are passed through to the OEM implementation. The OEM implementation should use its own methods to extract the information it needs from these NativeTypes, for instance the OEM implementation will need to provide a function to retrieve the dimensions of the pixmap or window through its own method.

The OEM must create an <egltypes.h> include file that typedefs the Native Types and provide this to the application writer alongside the standard Khronos include files.

2.4. Loading WS Library

As mentioned above, the WSEGL library is a dynamically loaded library. The WSEGL library is loaded by the EGL component of the OpenGLES driver, when the first eglGetDisplay() call is received. The code is loaded by following a set of rules to determine which WSEGL library is loaded. e.g.:

- Per native application settings stored in a file – determined by process name, or system wide default
- Via names and heuristics that are hard-coded into the OpenGLES driver (EG. If the X Server is running then use libpvrX11WSEGL.so)

In most cases the WSEGL library that OEMs have implemented will be loaded using the setting stored in a file as noted in the first method above.

Note: If all the APIs described in this document are not present in the WSEGL, then the OpenGL-ES driver will fail to load the module.

2.5. OpenGL ES and WSEGL Functions

While some WSEGL functions have implied counterparts to OpenGL ES EGL functions, an OEMs implementation of the WSEGL library should not presume that there will be a 1 to 1 correspondence between egl*() and gl*() functions and WSEGL_*() functions.

2.6. HW Prerequisites

Hardware rendering to a surface is restricted to a set of pixel formats, with certain conditions on the stride and address of that surface.

2.6.1. Pixel formats:

The pixel format of the surface must be one of the following:

- RGB 565

Bits	31:27	26:21	20:16	15:11	10:5	4:0
Component	R1	G1	B1	R0	G0	B0

- ARGB 4444

Bits	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
Component	A1	R1	G1	B1	A0	R0	G0	B0

- ARGB 1555

Bits	31	30:26	25:21	20:16	15	14:10	9:5	4:0
Component	A1	R1	G1	B1	A0	R0	G0	B0

- ARGB 8888

Bits	31:24	23:16	15:8	7:0
Component	A0	R0	G0	B0

- ABGR 8888

Bits	31:24	23:16	15:8	7:0
Component	A0	B0	G0	R0

2.6.2. Stride

The stride of the surface must be a multiple of 32 pixels. Stride differs from width in that the stride is the width rounded up to meet granularity requirements needed by most hardware.

2.6.3. Alignment

The surface address must be 4 byte aligned.

2.6.4. SGX MMU mapping

The surface must fit within the available SGX MMU slots. The MMU has enough slots to cover 256MB of memory and enough range to map memory over 4GB. Each MMU slot maps a 4k byte page, which is aligned to the platform’s 4k page. As the MMU is also used to map texture and parameter memory, not all the slots are available to map render target surfaces.

This render surface will be mapped in to the MMU via one or several function calls to PVR2D. These calls can either allocate and wrap a piece of memory, or just wrap a previously allocated piece of memory; when wrapping the memory it can be either contiguous or not.

To allow memory to be wrapped the OEM implementation must provide the physical page address of the underlying memory, and ensure that the virtual to physical mapping has been locked.

3. API Feature Walkthrough

3.1. Displays

An OEM implementation may support multiple displays. The displays are identified to the initialization function via a NativeDisplayType parameter. This is the parameter that is passed to the eglGetDisplay function. These will all run the same window system porting functions, but it will be possible to offer different display capabilities from within the same porting functions if the OEM implementation so chooses by identifying the NativeDisplay.

3.1.1. Validation

The NativeDisplay is passed in to the eglGetDisplay function. This value is then passed to WSEGL_IsDisplayValid. The validity check should be lightweight, it does not need to ensure the NativeDisplay is openable, but only that the display is valid.

The default display is always considered valid, and WSEGL_IsDisplayValid must always succeed for the default display.

3.1.2. Initialisation

Initialisation of the display may be a heavyweight function that allocates resources. If the default display token is passed in it may be necessary for the initialization function to establish a connection with the native display (for instance the X11 sample implementation is required to do this).

3.1.3. Configurations

Khronos EGL supports three render targets; these are Window, Pixmap and Pbuffer. Window and Pixmap support is supplied by the OEM implementation. The OpenGL-ES driver supplies Pbuffer support. When a display is initialized the supported configurations for Windows and Pixmaps are provided to the OpenGL-ES driver through the display initialization function. These configurations are used to derive EGL configurations to present to the calling application.

For an implementation to be conformant Native Window rendering targets must be supported.

The additional information present in the configurations is used to fill out the equivalent information in the exported EGL configurations. If a platform does not have specific use for this information it should be set to zero.

3.1.4. Capabilities

The capabilities reported by an OEM implementation allow the overlying OpenGL-ES driver to modify its behaviour with respect to the calls into the OEM implementation. Two sets of capabilities are currently supported, those related to swap interval, and those related to 2D/3D HW overlap.

3.2. Windows

A window is a back-buffered entity, with the contents of the back buffer made visible through the WSEGL_SwapBuffers call. Because of this they are straightforward to support. The back buffer can always be created to fulfill the HW pre-requisites described in section 2.

3.3. Pixmaps

Pixmaps are drawables where the SGX HW must render to the memory that has already been allocated by the window system for the pixmap. To make this possible the NativePixmap must have created with a priori knowledge of the HW pre-requisites. Alternatively at some suitable point in the WSEGL module the pixmap can be re-allocated to fulfill the HW pre-requisites. The choice of the behaviour is implementation dependant.

3.4. Synchronization and Parallelisation

Application level synchronization can be performed with either eglWaitGL() and eglWaitNative().

The WSEGL module can achieve HW parallelisation between the 3D and 2D by using PVR2D to perform data copies such as the presentation blit in SwapDrawable.

3.5. Drawable Parameters

The drawable parameters constitute the information required by the core OpenGL-ES driver to operate. They are used both by the SGX HW and the driver itself. Once the parameters for a drawable have been returned, they must remain valid until the drawable is been destroyed. If the WSEGL module needs to indicate to the OpenGL-ES driver that the drawable has in some way become incorrect (i.e. the overlying window the drawable is mapped to has changed size), then WSEGL_BAD_DRAWABLE should be returned from the GetDrawableParameters call, and the OpenGL-ES driver will call to destroy and re-create the drawable, ensuring that any outstanding HW activity has finished before calling destroy.

If the implementation supports Pixmap rendering (which is inherently single buffered), it should take great care to ensure that the overlying window system is aware that the SGX HW is rendering to these surface and only delete the surface after destroy drawable has been called.

The separate source and render parameters allow the WSEGL implementation scope for optimization in areas such as filling the render with a pre-generated background, our fulfilling progressive rendering semantics with more than one back buffer (and gaining better 2D/3D overlap).

4. Linux Sample Implementations

Two sample implementations are provided under Linux. One is a simple test mode implementation called Null Window System (NullWS). The second is an X11 implementation. These samples are described in the following sections.

4.1.1. Building

The linux sample implementations are built from the makefile found in the platform build directories i.e. `/embedded/build/<platform>`. The make target is opk i.e.

```
> make opk
```

This will build both the X11 and NullWS samples.

4.1.2. Library Loading

The driver heuristics for library loading come into effect when eglGetDisplay is called. The heuristics load the modules in turn (using dlopen()) and call WSEGL_IsDisplayValid until one returns that the display is valid. Modules are looked for in `/lib:/usr/lib:$LD_LIBRARY_PATH`.

The driver heuristics for library loading under Linux are: Open the module in the specified in the powervr.ini file which resides in /etc. If no file is specified then libpvrX11WSEGL.so will be loaded. If this does not validate the display; then libpvrNULLWSEGL.so will be loaded and queried for display validation. If this fails EGL_NO_DISPLAY is returned to the application. Note that having a DISPLAY environment variable set when X11 is not running will slow the heuristic sequence noticeably.

If a module is not found on the search path, dlopen() will fail and the next module in the heuristic will be attempted.

The powervr.ini file must be in Unix file format, and the structure of the file is:

```
[<AppName>]
WindowSystem=<filename>
```

where `<AppName>` can be the name of an application (in which case the setting only applies to the application) or `default` in which case the setting applies to all cases except where it is overridden by an application specific setting.

`<Filename>` is the full filename of the module to load i.e. `libpvrNULLWSEGL.so`

So an example where the PowerVR X11 module is the system default, and an OEM module is used for an application would look like:

```
[default]
WindowSystem=libpvrX11WSEGL.so

[JVM.exe]
WindowSystem=libOEMWSEGL.so
```

4.2. Null Window System

The Null Window System (NullWS) example is essentially a OpenGL-ES driver test mode that is offered here as the simplest possible WSEGL implementation under Linux. It is called NullWS for the reason that NULLs or zeros are passed in as the values of the Native Types. NullWS allows a single window drawable to be created to the full size on the default display.

4.2.1. Native Types

The Native Types used under NullWS are:

```
typedef void *NativeDisplayType
typedef void *NativeWindowType
typedef void *NativePixmapType
```

4.2.2. Features

The NullWS sample demonstrates the following:

- Retrieving the physical and logical address of the frame buffer from the operating system

- Wrapping the physical address using the PVR2DMemWrap function.
- Stride calculation of the back-buffer(s) based on the size of the screen
- Using PVR2DMemAlloc to create a back buffer
- Using PVR2DBlt to copy the back buffer to the visible buffer using a SGX accelerated bitblit
- Use of multiple back buffers to increase performance
- Using PVR2D to perform an asynchronous presentation blit
- Use of different source and render buffers to improve performance, while also retaining progressive rendering semantics

The NullWS sample does not demonstrate pixmap rendering as there is no mechanism to retrieve any properties of an implied pixmap, as only NULL would be passed as the value of the NativePixmap.

4.2.3. Display Implementation

The display in the NullWS code is implemented using the contiguous memory created by the /dev/fb device. The physical base is obtained from the OS using the FBIO_GET_FSCREENINFO ioctl(), and the dimensions and pixel format are obtained using the FBIO_GET_VSCREENINFO ioctl().

The logical address is obtained by calling mmap() for the /dev/fb memory.

The dimensions of the screen are used when creating the single allowed window drawable in this mode.

The pixel format is used to return the single supported config – i.e. windowed drawable with a pixel format that matches that of the /dev/fb device.

The display capabilities that are returned indicate that the WSEGL implementation will use SGX to control synchronization of the presentation blit, thus allowing better overlap between 2D and 3D hardware.

4.2.4. Window Drawable Implementation

The NullWS sample allows the creation of a single screen sized window drawable. The code itself doesn’t enforce the singularity, but relies on the calling EGL code to do so. The EGL code bases this on the uniqueness of the input value of the NativeWindow, and as there is only one valid value, i.e. NULL, there can only be one window creation request.

The NullWS sample includes conditional compliable support for multiple back-buffers. This feature in conjunction with using PVR2D allows increased performance through overlap of the 2D and 3D operations. By using the previous frames back-buffer as the source drawable in the GetDrawableParameters function this increased performance can be achieved while retaining progressive rendering semantics, and not require an extra copy of the back buffer to do so.

Back-buffers are allocated using the PVR2DMemAlloc function, which creates PVR2D surfaces that are subsequently blittable using SGX. When the back buffers are allocated the stride is calculated based on the width of the screen rounded up to the nearest 8 pixels.

4.2.5. Pixmap Drawable Implementation

The NullWS sample does not implement support for pixmap drawables, and so an empty function is provided. However some illustrative code is included but commented out. This code will not compile as it relies on such things as a pseudo function to interrogate the information about the native pixmap.

4.2.6. Copying Functions

As the NullWS does not support NativePixmaps neither CopyFromDrawable nor CopyFromPBuffer are supported, and both functions are empty. However as with pixmap drawables, some commented and non-compilable illustrative code is provided.

4.2.7. Swap Functions

The NullWS sample does not implement swap interval, so this function is implemented as an empty function.

The SwapDrawable function is implemented to use PVR2D to achieve overlap between 3D and 2D when the blit occurs. To ensure this performance benefit occurs the InitialiseDisplay function must report the WSEGL_CAP_WINDOWS_USE_HW_SYNC as TRUE.

4.3. X11 Window System

The X11 implementation uses standard X11 calls to interrogate information about the Native Types. It uses native types as defined by the Khronos Group. The X11 implementation will run on either an accelerated or un-accelerated Xserver (i.e. Xfbdev or Xmbx).

4.3.1. Native Types

The Native Types used under X11 are:

```
typedef Display *NativeDisplayType
typedef Window NativeWindowType
typedef Pixmap NativePixmapType
```

4.3.2. Features

The X11WS sample demonstrates the following:

- Retrieving the physical and logical address of the frame buffer by using the PVR2D frame buffer functions.
- Using PVR2DMemAlloc to create back buffers for Native Window drawables
- Using PVR2DBlt to copy the back buffer to the visible buffer using a SGX accelerated bitblit
- Use of multiple back buffers to increase performance
- Using PVR2D to perform an asynchronous presentation blit
- Use of different source and render buffers to improve performance, while also retaining progressive rendering semantics
- Using X11 functionality to generate the clip list for window blitting.
- Using X11 functionality to implement CopyFromPBuffer and CopyFromDrawable

The X11 sample does not demonstrate pixmap rendering as there is no mechanism in X11 to obtain the address of the memory being used for the Pixmap pixel data.

4.3.3. Display Implementation

Display validation only validates displays on the local host (i.e. a display name starting with ':'). If the default display is passed in then XOpenDisplay is called with NULL to generate a display to validate with the above test. If a display is opened in this function it will be closed before returning.

Likewise in initialize display, if the default display token is passed in, the default X11 display is opened, again by calling XOpenDisplay will the argument NULL.

The X11 sample expose a single config, which is for a window drawable at the same color depth as the display. However if the display color depth does not match a format supported by SGX no configs will be exposed.

The X11 sample exposes the WSEGL_CAP_WINDOWS_USE_HW_SYNC indicating that PVR2D will be used for blitting an so 2D/3D overlap will occur.

The following illustrative code indicates the relationship between the X11 Display, the EGL NativeDisplayType and the WSEGL NativeDisplay:

```
/*
// Application
*/
#include <X11/X11.h>
#include <GLES/gl.h>
#include <GLES/egl.h>

/* The EGL native display in this case is a pointer to an X11 Display */
typedef Display *NativeDisplayType

main()
{
        Display *dpy;
        EGLDisplay eglDisplay;

        /* Create the XWINdows display object */
        *dpy = XOpenDisplay(NULL);

        /* Send Display Object Native Type to EGL */
        eglDisplay = eglGetDisplay(dpy);
}
----------------------------------------------------------------
/*
// OpenGL-ES Driver
*/
#include <GLES/gl.h>
#include <GLES/egl.h>
#include "wsegl.h"

/* The EGL native display in this case is a anonymous pointer */
typedef void *NativeDisplayType

EGLDisplay eglGetDisplay(NativeDisplayType display_id)
{
   ...

   WSEGL_IsDisplayValid(display_id);

   ...
}
----------------------------------------------------------------
/*
// WSEGL Module
*/
#include <X11/X11.h>
#include "wsegl.h"

/* The EGL native display in this case is a pointer to an X11 Display */
typedef Display *NativeDisplayType

WSEGL_ERROR WSEGL_IsDisplayValid(NativeDisplayType hNativeDisplay)
{
        char *DisplayName;
        Display *dpy;

        dpy = hNativeDisplay;
        DisplayName = DisplayString(dpy);
        ...
}
```

4.3.4. Window Drawable Implementation

The X11 sample supports multiple window drawables within a process.Window drawables are backbuffered. Multiple back buffers are a compile time option. Multiple back buffers have a performance advantage by achieving better 2D/3D HW parallelisation without breaking progressive rendering semantics.

A windowed drawable will only be created if the window has the same color format as the display.

The back buffers will have a stride that matches the HW pre-requisite (i.e. 8 pixel granularity), even if the window width does note match this granularity.

4.3.5. Pixmap Drawable Implementation

There is no pixmap drawable support in this implementation, as no mechanism exists through standard X11 calls to recover the necessary information about the pixel memory underlying the Pixmap required by the 3D hardware.

4.3.6. Copying Functions

Both copy functions are implemented using standard X11 functionality. In both cases the source data is wrapped as an Ximage object, and the using the XPutImage function used to copy to the Native Pixmap.

Pbuffers are reversed in the Y direction compared to X11 pixmaps and the sample implementation illustrates the functionality required to flip in the Y direction as part of the copy.

In neither case is scaling or color conversion supported on the copy.

4.3.7. Swap Functions

Swaps are carried out using PVR2DBlit. As the implementation exports the WSEGL_CAP_WINDOWS_USE_HW_SYNC capability, swaps will benefit from 2D/3D hardware overlap. Swaps will be clipped to the visible portion of window on the screen. The clip list is calculated by traversing the window hierarchy using X11 function calls.

If the window size has changed between GetDrawableParameters and SwapDrawable the function will return BAD_DRAWABLE to the OpenGL-driver, and not carry out the swap. Implementors may choose to refine this behaviour, but should still return the BAD_DRAWABLE error if no swap is carried out.

5. Symbian Sample Implementations

Two sample implementations are provided for using with Symbian OS. One is a simple test mode implementation called which runs from the text shell analogous to the Linux Null Window System sample described previously. The second is an implementation that employs the Screen Driver and so runs under a UI. This should run under any Symbian UI (i.e. Series 60, UIQ), but has only been tested under TechView. These samples are described in the following sections.

5.1.1. Building

The build files for the OPK components are found in the `\embedded\opengles\ws\symbian` directory are built with the following commands.

```
> bldmake bldfiles
> abld build ARMV5
```

As with the Linux sample, this will build both the text shell and TechView samples.

5.2. Text Shell

The text shell is the name of the basic Symbian OS console, providing the simplest example of WSEGL. Again, as with the Linux sample, it supports only one drawable created with the same dimensions and pixel format as the display.

5.2.1. Native Types

The Native Types used in this example are:

```
typedef TAny *NativeDisplayType;
typedef TAny *NativeWindowType;
typedef TAny *NativePixmapType;
```

5.2.2. Implementation Details

The text shell sample was derived from the Linux version and, as such, is very similar. The pseudo code given for pixmap drawables is not reproduced and there is no equivalent of the USE_FBDEV method of obtaining the display memory. Other than minor changes in coding style, the differences are those necessitated by the Symbian development environment:

- The global display data is stored in the thread local storage (TLS) word.
- The Caps structure is not static as the Swap Interval values are written into it during initialization.
- Memory allocations use Symbian routines, rather than those in stdlib.

The default presentation method is to using flipping, by defining USE_PRESENT_BLIT blitting will be used instead.

5.3. Screen Driver (TechView)

The TechView sample builds on the text shell example in both design and functionality. Support is added for native pixmaps and rotated displays and the structure is much more object oriented.

5.3.1. Native Types

The Native Types used in this example are:

```
typedef TAny          *NativeDisplayType;
typedef RWindow       *NativeWindowType;
typedef CfbsBitmap    *NativePixmapType;
```

5.3.2. Drawables

A number of classes have been defined to simplify the handling of pixmap and window drawables as they share several attributes. The full list is given below:

```
class CColorBuffer : public CBase
```

Used to manage the construction and destruction of a buffer.

```
class CDrawableBase : public CBase
```

Abstract base class used for the common attributes of a drawable: size, format, rotation and the functions required to swap between buffers.

```
class CDrawablePixmap: public CDrawableBase
```

The pixmap drawable is the simplest drawable as it has only one buffer.

```
class CDrawableWindow: public CDrawableBase
```

A window drawable has multiple buffers that can be presented by either a flip or a blit. The method of presentation depends on the Window Rect (which defines the size and position of the window) and the Visible Regions (areas of the screen to draw to). If the window is the same size as the screen, then the buffers are flipped between, otherwise the contents of the last buffer are blitted to the screen.

```
class CDrawableDSAWindow : public CDrawableWindow, public MDirectScreenAccess
```

The Direct Screen Access window drawable is the same as the window drawable plus an implementation of the Symbian OS `MDirectScreenAccess` class. This inheritance requires us to implement the pure virtual functions Restart() and AbortNow(). Restart() is called when the drawing area changes and AbortNow() is the indication that drawing must cease.

```
class CBlitBase : public CBase
```

Abstract base class used to define the basic operations required to view a drawable, namely initialisation and presentation functions.

```
class BlitPVR2D : public BlitBase
```

An implementation of a buffer presentation system using PVR2D blitting. This method attempts to minimise the amount of effort in each presentation by pre-calculating and storing details about each blit. Firstly, when the drawable is created, the new BlitPVR2D object is based on the details that will never change about this drawable—namely size, rotation and the address and pixel format of the back buffers. The second optimisation comes from noticing that the areas of the screens to be drawn to will are constant. Therefore when the new drawing areas are given, these are converted into PVR2D blt commands ready to be submitted on every Swap().

5.3.3. Pixmap Support

The Symbian CfbsBitmap class is taken as the native type for pixmaps. The TechView sample demonstrates the creation of hardware accelerated pixmap drawables and the ability to convert a drawable or pbuffer to a CfbsBitmap with any display mode.

Rendering to a pixmap

The function WSEGL_CreatePixmapDrawable is simpler than WSEGL_CreateWindowDrawable, as there are no backbuffers to be created. The Symbian screendriver TAcceleratedBitmap… classes provide the functionality to get details about the pixmap (or rather CFbsBitmap) and verify that it's a hardware bitmap. An important point to note is that the `TAcceleratedBitmapInfo::iPhysicalAddress` member does not point to the physical address of the bitmap, as the Symbian documentation recommends, but to a PVR2DMEMINFO structure which we use to hold more information about the bitmap.

Copying to a pixmap

Support is provided to copy the contents of a drawable (window or hardware bitmap) or a pbuffer to a native pixmap (CFbsBitmap) with any display mode or rotation. In the generic case, routines in the CFbsBitmap class are used to copy and convert the pixels from the drawable or pbuffer to the pixmap. In a slight optimisation, the case where the drawable or pbuffer is the same format as the pixmap has been isolated to remove the format conversion. The situation where a drawable is copying to a hardware bitmap has been implemented using a PVRD2D blit—allowing for a hardware accelerated pixel copy that may also include format conversion.

6. WinCE Sample Implementations

'Null window system' and 'WinCE window system' sample implementations are provided for use with WinCE based operating systems. This sample is described in the following sections.

6.1.1. Building

The build files for these components are found in the `\eurasia\eurasiacon\wsegl` directory and built as part of the whole DDK as the DLL '`nullws.dll`'.

6.2. Null Window System

Three variations of the sample null window system driver are available:

- Blit based null window system
- Flip based null window system
- Front buffer rendering based null window system

6.3. WinCE Window System

The sample implementation is intended to work with the standard WinCE windowing system, itself driven by the GPE / GDI display driver.

6.4. Implementation Details

6.4.1. Native Types

```
typedef HDC             *NativeDisplayType;
typedef HWND            *NativeWindowType;
typedef HBITMAP         *NativePixmapType;
```

6.4.2. Features

The WinCE sample demonstrates the following:

- Retrieving PVR2DMEMINFO of the frame buffer by using the PVR2D frame buffer functions.
- Using PVR2DMemAlloc to create back buffers for Native Window drawables
- Use of multiple back buffers to increase performance
- Using WinCE functionality to generate the clip list for window blitting.

6.4.3. Display Implementation

This window system example only supports a single underlying display so only the default display is valid.

6.4.4. Window Drawable Implementation

The WinCE sample supports multiple window drawables within a process. Window drawables are backbuffered. Multiple back buffers have a performance advantage by achieving better 2D/3D HW parallelisation without breaking progressive rendering semantics.

A windowed drawable will only be created if the window has the same color format as the display.

The back buffers will have a stride that matches the HW pre-requisite (i.e. 8 pixel granularity), even if the window width does note match this granularity.

6.4.5. Swap Functions

Depending on which WinCE OS variant is in use, swaps are carried out using PVR2DPresentBlt, PVR2DPresentFlip or GDI BitBlt. Swaps will be clipped to the visible portion of window on the screen.

If the window size has changed between GetDrawableParameters and SwapDrawable the function will return BAD_DRAWABLE to the OpenGL-driver, and not carry out the swap. Implementers may choose to refine this behaviour, but should still return the BAD_DRAWABLE error if no swap is carried out.

7. Usage Examples

The following diagrams illustrate the important call sequences between modules in the cases of windowed rendering and pixmap rendering. Not all calls that would actually be made have been illustrated to keep these diagrams clear so that the important calls are not unduly hidden. These illustrative usage examples are independent of the operating system and window system.

7.1. Simple Windowed Rendering

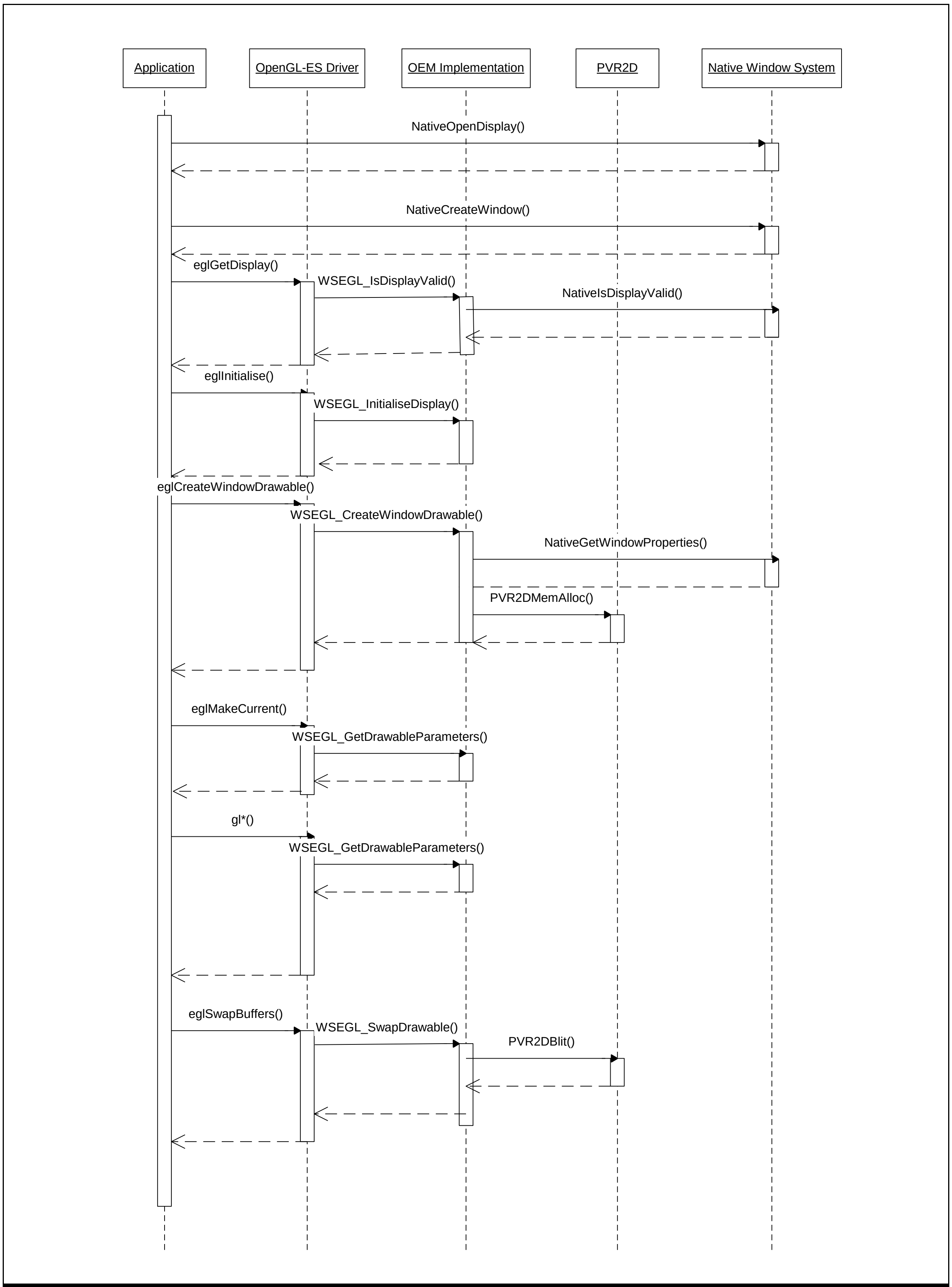


Figure 1 Simple Windowed Rendering

7.2. Simple Pixmap Rendering

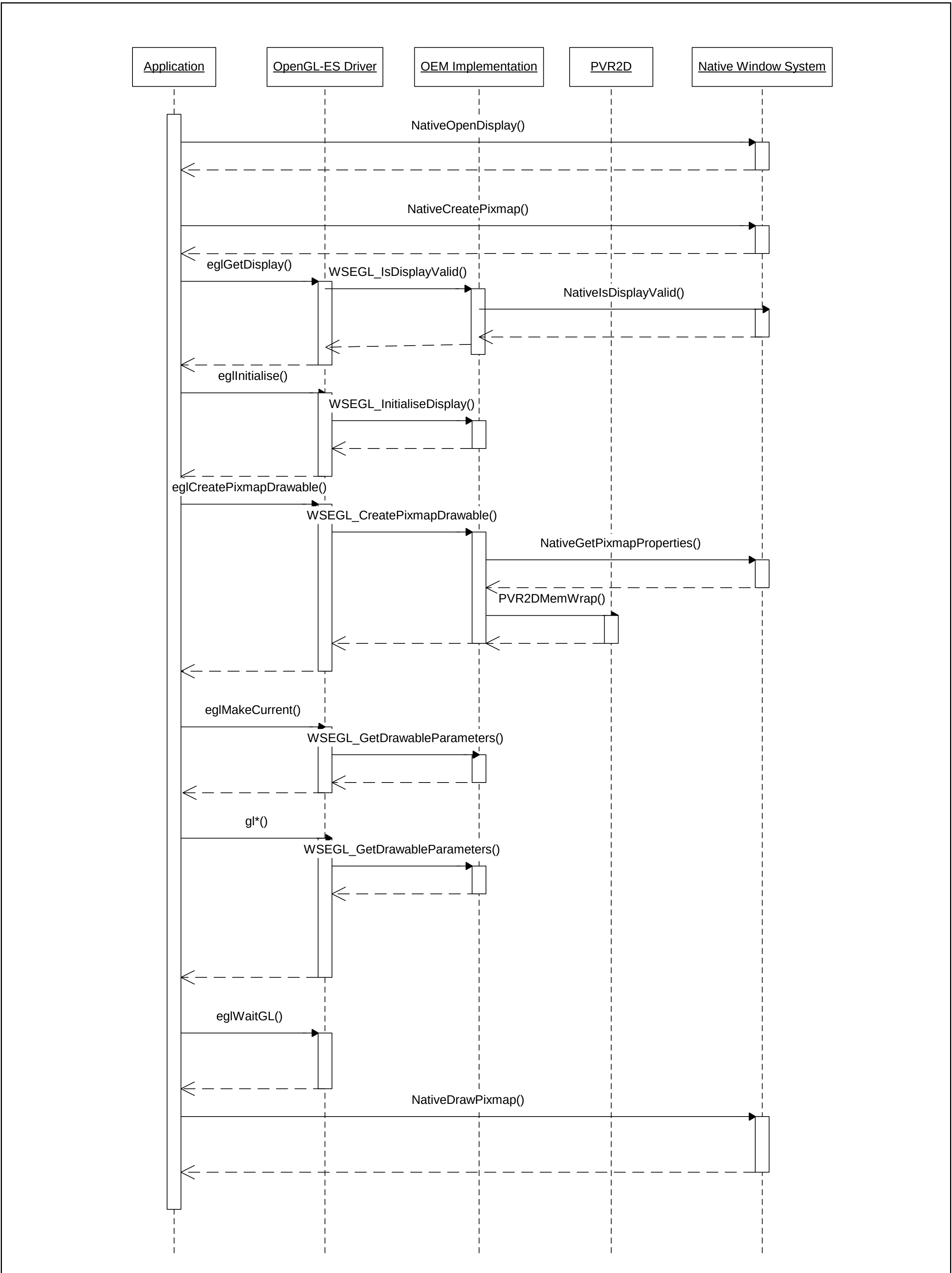


Figure 2 Simple Pixmap Rendering