SGL for Win95
SGL Direct Draw Programming Guide

Issue Number : 001

Issue Date : 10/9/96

Author : ssg

1. TABLE OF CONTENTS

2. PURPOSE AND SCOPE OF DOCUMENT

This document describes how the Windows 95 implementation of SGL can be used to provide a full screen 3d display, with optional 2d overlay support.

This will apply to both the separate and integrated PCI Bridge implementations of the MIDAS board (Midas 2 3 and 4), but does not apply to the stand-alone (separate screen) or overlay Midas 1 boards.

3. INTRODUCTION

SGL will be required to work with various types of graphics cards, some with advanced features and others with only basic features. To this end two modes of operation are provided to cater for both a generic interface and a low-level interface.
The amount of graphics memory, type of display card, and the user's preferred desktop mode will all have an effect on what can be done by SGL to optimise the 3d render process.

4. OVERVIEW OF SGL MODES

There are two basic modes of operation : direct draw mode, and address mode

SGL's Direct draw mode allows the application to leave all the buffer management to SGL. In this mode the application runs FULL-SCREEN 320x240, 640x400, 640x480 and 800x600 16 bit RGB; the application developer does not have to worry about page flipping or display memory management, it all happens automatically. SGL creates and controls the Direct Draw objects.

SGL's low-level address mode, on the other hand, allows the application to create and control the display buffers. SGL renders the 3d image at the start address defined by the application. This allows the **application** to implement various modes, including 3d-in-a-window. 3d-in-a-window requires the use of overlay and Blting techniques to avoid overwriting menus and dialog boxes.

SGL Mode :	SGL Direct Draw mode (page flipping)	SGL Address mode
Advantages	Can be fast even with a low cost graphics card. Needs less graphics memory. Large display area. Automatically changes to correct screen mode.	Can use menus, dialog boxes and other GUI facilities. Looks like a Windows App. All usual advantages of Windows over DOS. Can run alongside other applications, even those using Direct Draw.
Disadvantages	No GUI support, application must draw it's own controls. Looks like a DOS application.	Needs more memory and more powerful graphics system with overlay or fast Blt engine. User may need to change display mode for this to work. On-screen rendering suffers from cut-line. Clipping management is required by application.
Graphics Card Requirements	1 Meg of graphics memory for 320x240 and 640x400 image. 2 Meg for 640x480 and 800x600	2 Meg minimum, plus the ability to do overlay or transparent Blt.

5. 2-D SURFACE OPTIONS

There are at least two methods of allowing the application to draw a 2d image on top of the 3d : Overlay and Call-back. This allows the application to display text and graphics for uses such as cockpit data, maps, text messages etc.

2D Overlay allows the application to maintain a 2d image which is superimposed over the top of the 3d image using an overlay key-colour. The application must draw the 2d image in it's window client area with the key colour in areas required to be filled with the 3d image. 2D Overlay is only available in SGL Address Mode and is entirely the responsibility of the application.

2D Call-back allows the application to write into each 3d image buffer just before it becomes visible. The application can use GDI, Direct Draw, or write directly to the memory. 2D Call-back is only available in SGL's Direct Draw mode. (However, an end-of-render call-back is also available, see later on in this document for more details)

The following table shows the advantages of using the call-back and overlay methods of providing 2d .

SGL Mode	SGL Address Mode, overlay example. (Application controls Direct Draw)	SGL Direct Draw Mode with 2d call-back
Advantages	No speed penalty. App does not have to update 2d	No extra memory required. Compatible with all graphics

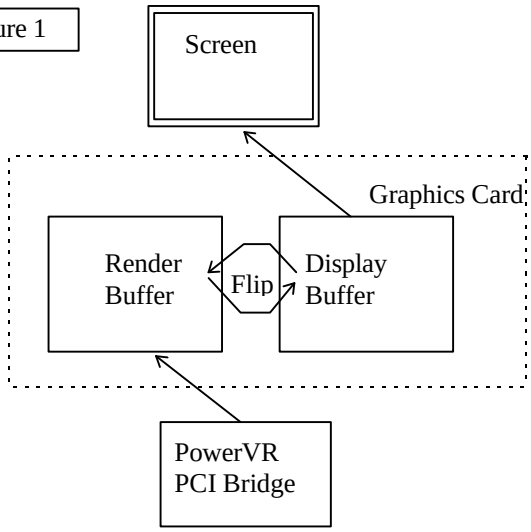
	surface for each frame. Allows windowed mode.	cards.
Disadvantages	Requires extra graphics memory for the 2d image. Requires a graphics card with hardware overlay capabilities or fast Blt engine	Speed penalty because the 2d image has to be re-drawn every frame.

It is also possible to draw 2d outside the viewport if the window client area is larger than the viewport. This area is not overwritten by the 3d image, and will provide a fast low-cost 2d option which will work with all graphics cards.

6. STANDARD SGL DIRECT DRAW MODE

This allows SGL to dynamically change display mode and take control of the entire display area, and uses a simple double buffered display mechanism as shown in figure 1 :

Figure 1



Two buffers are used, one is the on-screen display buffer and the other is the off-screen back-buffer. PowerVR renders to the back buffer while the front buffer is being displayed. When PowerVR has finished rendering the buffers are switched and the rendered image becomes visible. A third buffer may be used, if there is enough off-screen memory, to avoid having to wait for Vsync during the buffer switching process.

6.1. SGL Direct Draw Mode: Render sequence

1) App calls sgl_render()

2) SGL uploads the parameters to PowerVR and prepares as far as possible for the next frame.

3) SGL waits for PowerVR to finish rendering the previous frame, then calls back the application.

4) App can blt/draw 2d on top of previous frame.

5) App returns to SGL from the call-back.

6) SGL Flips previous frame to make it visible, and Locks the next frame buffer.

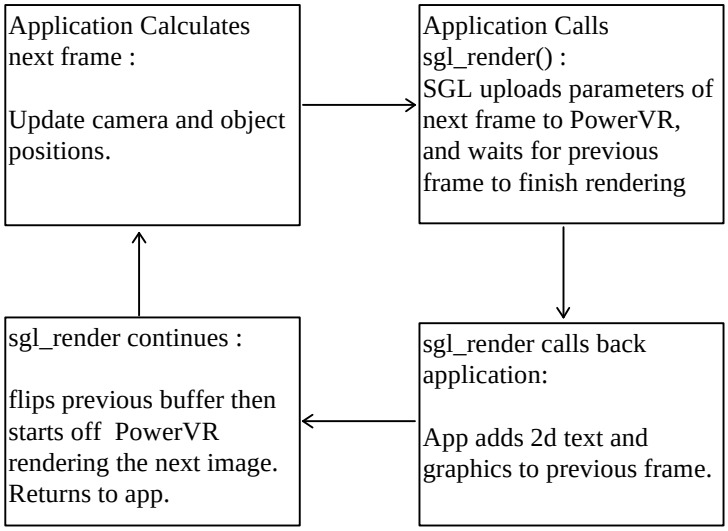
7) SGL converts logical to physical address and programmes the bridge.

8) SGL starts the hardware render process.

9) SGL returns from sgl_render() while PowerVR is rendering the image.

6.2.SGL Direct Draw Mode : Example Application Loop

The application's display loop is shown below :



6.3. Limits Imposed By Midas 3

The only RGB format supported by Midas 3 is RGB 565. However, both RGB 555 and RGB 565 will be supported by the PCX1 Midas 4 solution.

If the graphics card does not support RGB565 then please contact the graphics card manufacturer for a possible updated direct draw driver for that device. Images will render with a Graphics mode running in 555 but colours will be incorrect. Principal graphics controllers affected are:

ViRGE
Cirrus Logic
ATI
Trident

IMPORTANT: This limitation has been overcome for the Compaq Presario systems fitted with a ViRGE controller. However, to ensure correct operation, programmers must pay special attention to the load/unload sequence of SGL and DirectDraw libraries. Please see Appendix One.

6.4. Limits Imposed by Graphics Card

1 Meg of graphics memory is required to support 640x400x16 bit.
2 Meg of graphics memory is required to support 640x480x16 bit.
4 Meg of graphics memory is required to fully support 3d in a window.

6.5. SGL Direct Draw Mode : 2D Call-back

The application can request a call-back to allow it to draw a 2d image on top of the render buffer after the 3d image has been rendered but just before the page flip takes place. The app can use GDI to write text and graphics, or it can directly modify the buffer memory. This call-back is completely different from the AddressMode call-back mentioned in the next section.
See the API section at the end of this document for more details.

7. SGL ADDRESS MODE (APPLICATION CONTROLS DIRECT DRAW)

A low level mode is provided which allows the application to do the display buffer management. In this mode the application passes down the address of the render buffer, as well as the stride and pixel depth, and SGL will render into that area. The application can use Direct Draw or any other mechanism to provide the render buffers, and can then control the way that the image is made visible by using blting, overlay, or page flipping. This allows 3d-in-a-window applications to be written.

This allows SGL to be used to render into a buffer which can be displayed as a normal overlapped Window, and allows menus and dialogues to be used in the normal way.

7.1. SGL Address Mode: Render sequence

The sgl_render() internal sequence for SGL Address Mode is shown below:

1) App calls sgl_render()
1) SGL uploads the parameters to PowerVR and prepares as far as possible for the next frame.
2) SGL waits for PowerVR to finish rendering the previous frame, then calls back the application.
3) App modifies and displays the previously rendered buffer in any way it chooses (eg. flip)
4) App locks next render buffer and passes address back to SGL.
5) App returns to SGL from the call-back
6) SGL converts logical to physical address and programmes the bridge.
7) SGL starts the hardware render process.
8) SGL returns from sgl_render() while PowerVR is rendering the image.

Note : Exactly what happens during the call-back is up to the application, but it must return a pointer to the next buffer to SGL.

The call-back referred to in this sequence is the AddressMode call-back which is totally different from the 2dCallback referred to in the previous section.

7.2. Limits Imposed By Direct Draw and Display Driver

There is no known way of switching between an RGB565 and RGB555 desktop, which may effect an overlay key colour, for instance.

7.3. Limits Imposed By Graphics card

The graphics card must have a graphics accelerator capable of fast Blting or colour keyed overlay. Also it must have enough graphics memory for the main desktop area as well as two off-screen surfaces if overlay is being used.

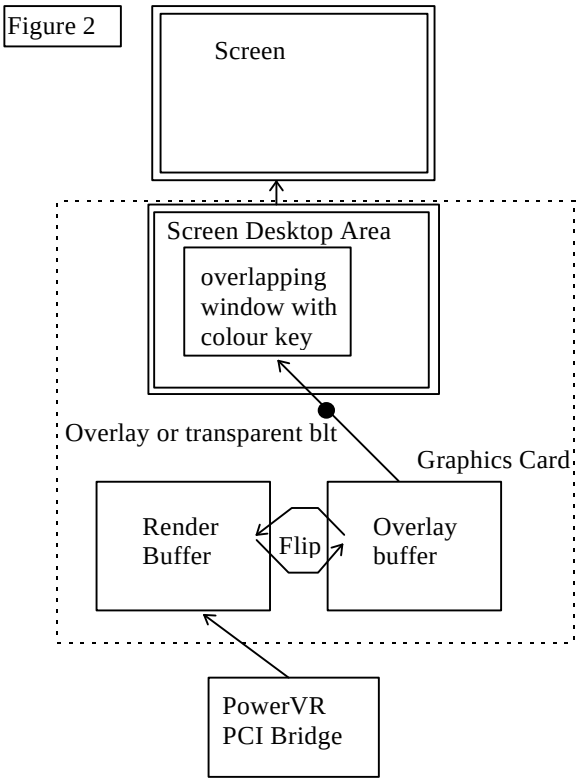
7.4. SGL Address Mode : 2D Overlay Example

The application must use overlay or colour keyed Blting to implement windowed mode. This allows 2d overlay and 3d-in-a-window, and also does the clipping of overlapping menus/dialogs etc.

Note that the graphics hardware and software must support RGB overlay for this to work, and in some configurations there may be video memory bandwidth limitations that will require the user to change to a lower screen refresh rate. In SGL's Address mode it is the application's responsibility to determine the hardware capabilities of the graphics device and to setup the necessary overlay/color-keyed surfaces

Figure 2 illustrates the mechanism which can be used to achieve the overlay :

Figure 2



The above diagram shows how an application can use SGL's Address mode to render into an off-screen overlay buffer. The application must create and manage the buffers.

7.5. SGL Address Mode : Blting of 3D Image Example

If the graphics system doesn't support colour keyed overlay or transparent Blting then the application can use conventional Blting, using a Direct Draw Clipper object to manage the overlapping windows, menus, and controls.

The 3d image is Blted into the window using the Direct Draw's clip list to avoid overwriting the obscured areas. The application must create and manage the buffers and clip-list objects.

8. COMMON SGL/DIRECTDRAW RELATED PROGRAMMING ERRORS

Some common programming errors relating to SGL and DirectDraw usage are:

1) The app must set swap_buffers to TRUE in sgl_render unless multiple viewports are being used.

2) The app's 2dcallback must return 0 unless a fatal error has occured.

3) The app must set the sgl_device_colour_types device_mode correctly in the sgl_create_device call to either 24 bit or 16 bit as required.

4) The app must set the bBitsPerPixel parameter in the CALLBACK_ADDRESS_PARAMS if using sgl address mode, to 15, 16, 24 or 32. Note that Midas3 only supports 16 and 32.

5) If using SGL in address mode to render to a 24-bit packed surface, note that the pixels are 3-byte aligned but SGL must start rendering on a 4-byte boundary, so the following rule must be applied:
The render address must be a multiple of 4 and a multiple of 3, ie on a 12 pixel boundary in X direction. If rendering starts at the beginning of a Direct Draw surface then this will happen automatically. See Tower5 sample code for an example of how an application uses the WM_WINDOWPOSCHANGING message to force the start address to be on a 12 pixel boundary. If this is not done then the wrong colours will result.

6) Applications must monitor the Windows WM_ACTIVEAPP message. Whenever the wParam is FALSE the application should ensure that no renders to SGL take place.

9. API EXTENSIONS

The Windows 95 specific extensions to SGL are described in the SGL API document, but are summarised here :

9.1. sgl_use_ddraw_mode

```
int sgl_use_ddraw_mode (HWND hWnd, PROC_2D_CALLBACK Proc2d);
```

This allows the application to pass down it's window handle when SGL's normal Direct Draw full-screen mode is being used. This function MUST be called before sgl_create_screen_device (unless address mode is being used in which case the function is never called). This function tells SGL that direct draw mode is being selected.

The application can pass down it's 2d call-back routine in Proc2d. SGL calls the call-back just before the 3d image is made visible. The application can then modify the 3d image in the call-back function as required using GDI or any other method to modify the image.

In order to improve performance, SGL returns from sgl_render before PowerVR has finished rendering, and so the rendered image does not get diplayed until the next call to sgl_render. The

2d callback is made during the next call to sgl_render, and so the application is in fact adding the 2d to the previous frame not the current frame. Normally this will have no effect, but it does explain why there is a one frame lag in the 2d callback. There are in fact three frames in the system, the frame in the front buffer that is visible, the frame in the back buffer which is having 2d added to it, and the frame that is yet to be rendered, which is represented by the current state of the application's object variables and SGL's display list.

SGL gives the 2d call-back the following parameters :

```
typedef struct
{
    LPDIRECTDRAW        pDDObject;
    LPDIRECTDRAWSURFACE p3DSurfaceObject;
    LPVOID              p3DSurfaceMemory;
    WORD                wBitsPerPixel;
    DWORD               dwStride;
    WORD                wWidth;
    WORD                wHeight;

}CALLBACK_SURFACE_PARAMS, *P_CALLBACK_SURFACE_ PARAMS;
```

Note : the call-back is optional; if no call-back is required then Proc2d should be set to 0.

9.2. sgl_use_address_mode

```
sgl_use_address_mode (PROC_ADDRESS_CALLBACK ProcNextAddress,
                      LPDWORD *pStatus);
```

This selects the low level address mode, and the application passes in a call-back which SGL will use to get the memory address of the render buffer. SGL contains a status DWORD which is modified at interrupt time to signal that PowerVR has finished rendering. SGL Passes the address of it's status dword back to the application by setting *pStatus to contain the address of the DWORD.

The application must provide the following information to SGL on return of the call-back :

```
typedef struct
{
    LPVOID  pMem;
    WORD    wStride;
    BYTE    bBitsPerPixel;

} CALLBACK_ADDRESS_PARAMS,*P_CALLBACK_ADDRESS_PARAMS;
```

In address mode the application must provide a call-back similar to the following example :

```
/* This function is called by SGL during sgl_render */
int ProcNextAddress(P_CALLBACK_ADDRESS_PARAMS pParamBlk)
{
    /* Make previous frame visible: unlock then flip or blt */

    /* Lock next buffer to get address */

    /* Give info back to SGL */
    pParamBlk ->pMem = pNextBuffer;
```

```
    pParamBlk ->wStride = wStride;
    pParamBlk -> bBitsPerPixel = 16;

    return sgl_no_err;
}
```

Note that pMem must be a valid logical pointer , i.e. the memory should be accessible in the normal way so that *(char*)pMem = 0 does not cause a GP fault ! The memory must be locked (i.e. should not physically move or be paged out to disk), and should be one physically contiguous DWORD aligned block[1]. The advised method is to use Direct Draw to allocate off-screen graphics memory.

[1] Note that in 24 bit packed mode the pixels are aligned on 3-byte boundaries, which is acceptable provided that the first pixel is on a DWORD boundary, ie the least significant two bits of pParamBlk ->pMem must be zero, otherwise the rendered image will have the wrong colours. See 8.5 for more details.

9.3. sgl_use_eor_callback

```
int sgl_use_eor_callback(PROC_END_OF_RENDER_CALLBACK
    ProcEOR);
```

This allows an application to give SGL the addrsss of a routine that is called as soon as the hardware has finished rendering the previous frame. The call-back is in fact called during an sgl_render command when SGL first detects that the previous frame has been rendered.

The End Of Render call-back was put it to allow the application to unlock the render buffer (address mode) and process any pending windows messages as soon as possible to prevent windows from locking up :
A side effect of locking a direct draw surface is that the Windows display freezes until the surface is unlocked; the direct draw documentation says :
“In order to prevent VRAM from being lost during access to a surface, DirectDraw holds the Win16 lock between **Lock** and **Unlock** operations. The Win16 lock is the critical section used to serialize access to GDI and USER. Although this technique allows direct access to video memory and prevents other applications from changing the mode during this access, it will stop Windows from running, so **Lock**/**Unlock** and **GetDC**/**ReleaseDC** periods should be kept short. Unfortunately, because Windows is stopped, GUI debuggers cannot be used in between **Lock**/**Unlock** and **GetDC**/**ReleaseDC** operations.“

sgl_use_eor is used in sample code to overcome this limitation.

In practice, most graphics cards do not need the buffer to be locked during render, and so this facility may not be needed for an OEM solution. However, in order to be compatible with all graphics cards strict direct draw locking must be observed (though this has a slight performance penalty).

It is strongly advisable to use the sample code in the PowerVR SDK as a basis for understanding these issues.

APPENDIX ONE: MIDAS 3 VIRGE COMPATIBILITY FOR COMPAQ PRESARIO

To ensure correct switching of display modes, certain general rules must be followed when using Midas 3 on the Compaq ViRGE systems. These are summarised as follows:

1) The application must not call Direct Draw until after SGL.DLL has been loaded. This allows SGL to get to Direct Draw before the app, to initialise the display hardware correctly.

2) If the application has changed the Direct Draw cooperate level to exclusive, then the application must close all it's windows or set its cooperative level back to NORMAL, BEFORE the direct draw object released and BEFORE the application closes down. This forces direct draw to remove the hWnd from it's exclusive mode list and allow SGL to close down and restore the desktop correctly.

3) If the application has created some direct draw surfaces, they must all be released and the direct draw object released BEFORE the application closes down.

General Programming sequence for SGL applications :

1) Load SGL (in order for it to initialise the display) before loading DirectDraw directly.

2) App loads Direct Draw if it needs to.

3) App does normal stuff until closedown.

4) App closes it's windows if Direct Draw exclusive mode has been set or restore window to NORMAL mode.

5) App closes DirectDraw.

6) App unloads SGL (and thus restore display).

In the SDK examples, Tower4 and Tower5, SGL is statically linked and will automatically get loaded on application startup and closedown.

In Tower4, DirectDraw is not used outside of SGL.

In Tower5, DirectDraw is loaded and unloaded from WINMAIN() in the module FRONTEND.C. DirectDraw is loaded through a call to DDRAWINIT(), in the module DDMEM.C, and is unloaded through a call to DDEND(), also in the module DDMEM.C