Rendition

The Vérité™ VLib API

(Excerpted Draft)

A Guide to Vérité™ Graphics Rendering

0. Document Change Record / Revision History

07-09-xx Alan M. Levinson: Engineering Technical Writer
version 0.6: completed initial documentation of the Pixel Engine State Management Functions. completed initial documentation of the Display Management functions; distinguishing V3300 and V4400-only funcs and variables, with emphasis on the V4400.

05-26-xx Alan M. Levinson: Engineering Technical Writer
version 0.5: implement marked revisions specified by Chet Haase as well as other changes resulting from our 5/20 doc review: put all public typedefs and structs in one section…
Note: **version 0.4** has been checked into cvs and added documentation for the individual Microcode Management Functions as well as for structs that were not doc'd up to then: v_cmdbuffer, v_verite ….

05-11-xx Alan M. Levinson: Engineering Technical Writer
version 0.3: complete initial documentation of the individual Context Management Functions and the corresponding discussion section "Vérité Context Management (Initialization and Shutdown)".

04-22-xx Alan M. Levinson: Engineering Technical Writer
version 0.2: complete initial documentation for each of the Graphic Primitive functions and the associated discussion section: "All About VLib's Graphic Primitives."

04-13-xx Alan M. Levinson: Engineering Technical Writer
version 0.1: first partial draft: it contains complete documentation of the Command Buffer Functions, i.e.each individual function is documented and two discussion-type sections (All About Command Buffers, Command Buffers -- Some Sample Code) are drafted. The Introduction and Overview sections are minimal but clean

Table of Contents

II. POINTERS TO THE DOCUMENTATION

Stylistic Conventions Used In This Document

This document uses a minimum of stylistic conventions to clarify information.

Here is an exemplary summary:

{nn,nn,nn}	All comma-separated values are valid but only one of the values can be specified.
Myfile.txt	Actual filenames appear in bold text. Bold text is also used for subheadings and to emphasize a word or phrase.
Section IV	Hyperlinks appear underlined.
VL_CreateCmdBuffer	Variable names, including public API functions like this, are italicized.

Additionally, note that each of the public API functions is described in the same format as the others. A summary of that format is presented at the start of the section which lists the functions, Section V**.**

Other Recommended Documents

This document refers to other documents as relevant. It is recommended that you have not only access to, but familiarity with, the following:

The V3300 Target Specification
Revision 1.7 is available internally.

The V4400e Target Specification

The Central Services 3.0 Specification: **./docs/central/cs3_0.doc**

The V3x00 3D Microcode Specification: **./docs/ucode.doc**

The V3x00 3D Microcode API Doc: **./docs/ucode/ucode3d3k.doc**

III. INTRODUCTION

Why VLib? What Functionality Does It Offer?

This document details the VLib Application Programmer Interface (API) for various releases of the Vérité chip. The V3300 and V4400 chips are specifically covered in this document, with emphasis on the V4400. The library provides access to the 2D and 3D microcode interfaces for the chips it supports. It is provided as a statically linked library.

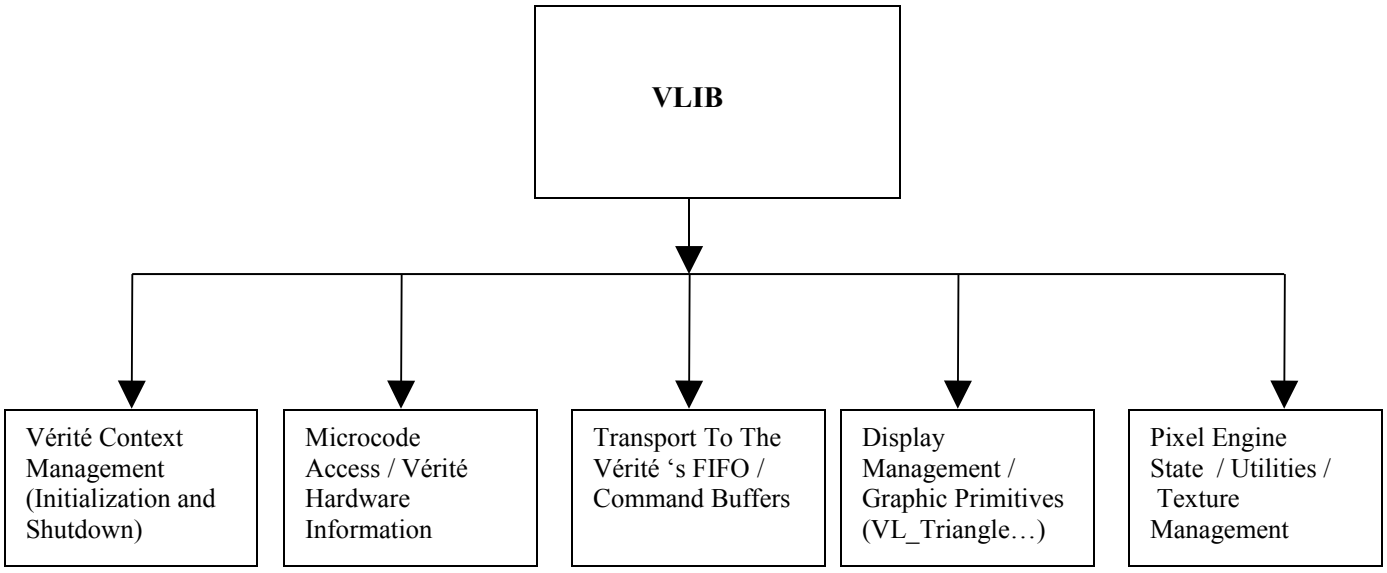
VLib supports three API drivers: OpenGL, Direct Draw and Direct3D. Some of the functions in the VLib API functions are applicable only to the 3D drivers, others only to the 2D API driver (Direct Draw).

VLib has a number of different facets. **The Components Of VLib** diagram (below) shows the five main elements: Vérité Context Management (Initialization and Shutdown), Microcode Access / Vérité Hardware Information, Transport / Command Buffers, Display Management / Graphic Primitives and, lastly, Pixel Engine State / Utilities / Texture Management.

How Is This Document Organized?

Each C function in the VLib API belongs to one of these five components. Section IV, which follows, presents an overview and discussion of each component.

Diagram: Components Of VLib



Section V is a complete listing of VLib's public functions. A description of each function, as well as of the arguments it expects, is provided. Section V-B lists and describes VLib's global constants, typedefs and structs. Appendix A offers some troubleshooting tips and Appendices B through D provide on-line references to information from the API vendors.

IV. A DETAILED DISCUSSION

A. Overview

As stated in the Introduction, the library has five major elements. The first two are straightforward. The first, Vérité Context Management, provides the set of functions which accomplish the tasks of setting up, maintaining and closing out, connections to the Vérité. The second, Microcode Access / Vérité Hardware Information, provides a way to execute 2D and 3D microcode rendering functions on the Vérité. Both are discussed below (Vérité Context Management (Initialization and Shutdown) and Microcode Access / Vérité Hardware Information).

The other three key elements (Command Buffers, Pixel Engine State / Utilities / Texture Management and Display Management / Graphic Primitives) all offer abstractions for ease of programming the Vérité.

First, why Command Buffers? The most optimal way to send data to the processor is to package command and texture data into arrays and issue DMA commands to transfer these data collections to the Vérité. The command buffer abstracts the management of these arrays of commands and is discussed in detail below (All About Command Buffers).

The next key element is Pixel Engine State / Utilities / Texture Management. Textures are built on Surfaces and the Pixel Engine state is critical to both. Pixel Engine State Management is discussed below, under About The Pixel Engine State Management Functions.

A Surface is an abstraction (structure) which describes a rectangular collection of pixels. Surface attributes describe both the organization of the rectangle (width, height, byte stride, address), as well as the interpretation of individual pixels (number of buffers, pixel depth, format, chromakey). The VLib Surface functions translate Surface descriptions into Vérité State Management functions, thus simplifying the management of the Vérité rendering state. This abstraction is discussed in detail below (All About Surfaces).

Lastly there are the Display Management functions and Graphic Primitives. The former are discussed in About The Display Management Functions. The Graphic Primitives, like *VL_Triangle*, package arrays of microcode rendering function calls and their associated data into a command buffer. The additional function call they introduce makes them inefficient. As is explained below, in All About VLib's Graphic Primitives, it is faster to use command buffers to invoke the microcode functions themselves.

Before looking at the diverse functionalities of VLib, it is important to first understand the role it plays in relation to the Vérité hardware, the graphic APIs and another critical library, Central Services. This is the subject of the next subsection, VLib In Relation To Its Neighbors.

B. VLib In Relation To Its Neighbors

The diagram on the following page presents the relationship between VLib, the three API drivers (Direct Draw, Direct3D, OpenGL), the Central Services library and the Vérité hardware. As you can see, the drivers are clients of VLib and VLib is a client of Central Services. Each API driver is also a client of Central Services. Note that only Central Services talks directly to the Vérité / hardware.

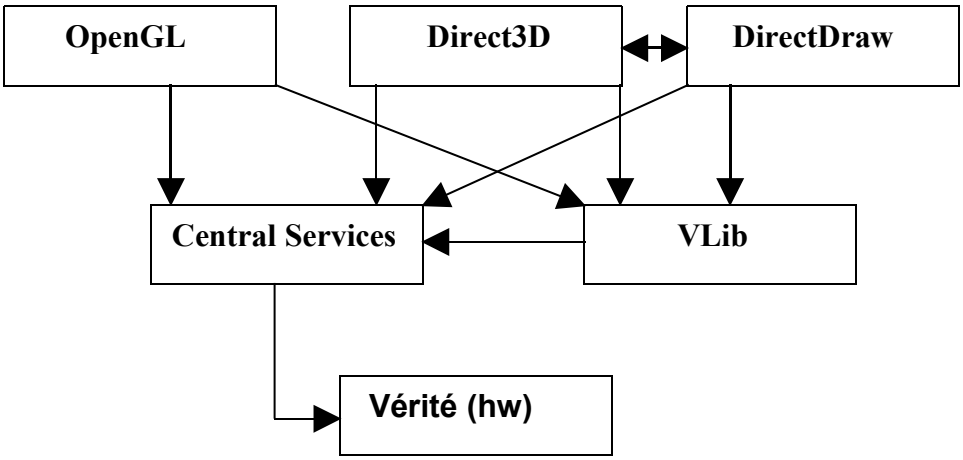
Since VLib talks to Central Services, and not to the hardware directly, a brief overview of the role Central Services plays will make it easier, and clearer, to focus on what VLib does. Central Services is the Vérité hardware abstraction layer and implements a set of generic routines that can be used to access multiple or specific Vérité devices.

Central Services provides a number of different functionalities: 1) Context Management 2) DMA Register Access 3) Microcode Loading and Export Table Management 4) Mode and Display Management 5) Vérité Framebuffer Memory Management 6) Allocation of Physically Locked Memory 7) Allocation and Management of AGP Memory 8) Framebuffer and MMIO Memory Space Mapping.

VLib's goals are not unrelated – VLib also provides access to the Vérité microcode as well as context and display management functions. So why two libraries? The fact is the two libraries provide similar functionalities to different types of clients. However, only VLib provides access to the 3D microcode so Direct3D and OpenGL call VLib to do 3D rendering. The command buffer abstraction is also provided only by VLib.

Why does VLib call Central Services? Only Central Services has direct access to physically locked memory so VLib needs to call Central Services to get memory it can then allocate to its (3D) clients. GDI, Video and the 2D DirectDraw API access memory directly though Central Services itself.

Diagram: VLib In Relation To Its Neighbors



C. Vérité Context Management (Initialization and Shutdown)

As the section heading suggests, the functions comprising the Context Management element of VLib perform **Initialization** and **Shutdown** tasks associated with a Vérité device. There are also three other facets of Context Management with which specific functions are associated: **Locking / Unlocking Pages**, **Allocating/Freeing Memory** and **Adding / Removing Microcode Contexts**.

All are discussed, at least briefly, below. The individual functions mentioned in this section are documented in Section V, under the Microcode Context Functions.

Initialization

Peter? Chet: Is the crtc stuff changing (> two crtc bases.. 'a per csdisplay.doc and CentralServices4) to deal with new inputs…? And any better words than 'filling in' and 'crtc bases' (which are from Affie's vlib.doc)?

The initialization of a Vérité device is done by calling `V_OpenVeriteLibrary`. This function calls private functions which accomplish three critical tasks: the filling in of the addresses of the two crtc bases used to set the appropriate display base, the filling in of the texture spr definition table for the two sources supported by the Vérité, and the initialization of the microcode export table structures.

Microcode Accessibility

This last task makes the microcode function addresses available to VLib clients – i.e. it makes it possible to call the 2D and 3D microcode procedures from VLib. The information is stored in the public structs `vl_d3d_vertexProcs, vl_primitiveProcs, vl_primitiveProcs2d`, `vl_renderProcs` and `vl_vertexProcs`.

Each member of these structs is named after a microcode function and contains the 32-bit address of that microcode function. These addresses are used by VLib to add microcode command blocks into the command buffer. They also allow a client to directly add 3D microcode command blocks into the command buffer or, for that matter, to invoke the microcode, as DirectDraw sometimes does, without using command buffers whatsoever.

The Graphic Primitive functions (`VL_Triangle` ..), the Microcode Management functions (`VL_GetNumRenderProcs` ..) and the Pixel Engine State Setting functions (`VL_SetALUMode` ..) all explicitly access the microcode through these variables. See VL_Triangle: A Look At A Few Lines Of Code for an example.

Multiple Microcode Contexts

`V_OpenVeriteLibrary` needs a microcode context handle since it needs to get information on exported microcode addresses. A single `v_verite` object should be created for all Vérité contexts of a process i.e *V_OpenVeriteLibrary* should be called once per process. This allows applications to share command buffers for all contexts of a process. If multiple microcode

contexts intend to share command buffers, then `V_OpenVeriteLibrary` should be called only when the first microcode context is created. (Call `VL_AddUcHandle` to add additional contexts.)

The function returns a handle to a Vérité device. Many VLib functions require this handle as one of their arguments, or use it as a pointer to a command buffer.

Shutdown

When an application terminates, or an error occurs, the Vérité needs to be shutdown properly. This involves destroying the command buffers, destroying the front and back buffer, and closing the handle to the Vérité itself. For sake of example, let's say this is done within the function `myApp_Close`. This function is called after the main loop exits, or by an error handler when an error occurs.

To destroy each of the command buffers in the ring, use the function *VL_DestroyCmdBuffer*. This function will wait until all issued commands on the buffer are complete and then free the memory associated with the buffers:

```
void myApp_Close ( void )
{
    int  i;

    // free command buffers if they exist
    if  (cmdbuffer)
    for (i=0; i<myApp_NUMCMDBUFS; i++)
        VL_DestroyCmdBuffer(cmdbufs[i]);
```

….

THIS PAGE IS THE MARKER BETWEEN THE FIRST SECTIONS OF THIS EXCERPTED SPEC AND THE LATER SECTIONS, WHICH FOLLOW.

….

Name

VL_SetZScissorEnable

C SPECIFICATION

void VL_SetZScissorEnable(v_cmdbuffer **cmdbuffer, int scissor_z);

PARAMETERS

cmdbuffer	*Pointer to the active command buffer*
scissor_z	*Flag for Z scissor enable*

DESCRIPTION

This function sets the pixel engine ZScissorEnable register. If `scissor_z` is set, pixels with a Z coordinate that overflows the valid range are not drawn. A Z value overflows if:

(Z & 0xF8000000) != 0

If `scissor_z` is not set, overflows of the Z coordinate are ignored, so no pixels are rejected on this basis. To fully clear the hardware state, a Z value must be set after disabling ZScissorEnable. The internal Z overflow state is only updated each time a Z value is written.

The ZScissorEnable state applies to drawing operations but not to bit blit operations.

NOTES

1) The ZScissorEnable state functions independently of other Z buffer state. Pixels which overflow in Z can be rejected even when a Z buffer is not being used.

2) The source for this function adds a `vl_primitiveProcs.prim_SetPEState` microcode command block into the command buffer using `VL_AddToCmdList`.

SEE ALSO

About The Pixel Engine State Management Functions

`VL_SetZBufWrMode`

RETURNS

NAME

VL_SetZStride

C SPECIFICATION

void VL_SetZStride(v_cmdbuffer **cmdbuffer, v_u32 zStride);

PARAMETERS

cmdbuffer	*Pointer to the active command buffer*
zStride	*Stride value for the Z buffer*

DESCRIPTION

This function sets the pixel engine ZStride register to the specified hardware stride value *`zStride`*. This stride value is used together with the ZBase state to address pixels in the Z buffer for drawing operations when Z reading or writing is enabled.

The board address of a Z value at location (X, Y) in the Z buffer is given by:

$$\text{ZBase} + (Y \times \text{V\_StrideToLinebytes(ZStride)}) + (X \times 2).$$

NOTES

1) Stride values are determined by the particular hardware in use and, hence, should always be obtained explicitly through VLib calls.

2) The source for this function adds a *`vl_primitiveProcs.prim_SetPEState`* microcode command block into the command buffer using *`VL_AddToCmdList`*.

SEE ALSO

About The Pixel Engine State Management Functions

`V_LinebytesToStride`, *`V_StrideToLinebytes`*, *`VL_SetZBase`*, *`VL_SetZBufWrMode`*, *`VL_SetZScissorEnable`*

RETURNS

K. STRIDE FUNCTIONS

```
v_u32       V_LinebytesToStride(v_u32 lb, v_u32 *realLb);

v_u32       V_StrideToLinebytes(v_u32 strideReg);
```

L. SURFACE MANAGEMENT FUNCTIONS

```
v_pixel_buffer    *VL_CreateDrawBuffers(v_verite *verite,
                    v_pixelbuffer_alloc_desc *pixelDesc);

int               VL_DestroyDrawBuffers(v_verite *verite,
                              v_pixel_buffer *pixBuffer);
```

M. TEXTURE MANAGEMENT FUNCTIONS

See vtex.h …

VI. APPENDICES

Appendix A – Troubleshooting Tips

Unsure how much use we will make of this section...

We might want a "SYSTEM HANGS" Symptom/Response thing?

The V3 3D API Microcode Doc (./docs/ucode/ucode3d3k.doc) has a fairly lengthy Troubleshooting Section, check it out for parallels …?)

Appendix B - OpenGL Vendor Reference

The public OpenGL website is located at http://www.opengl.org.

Appendix C - Direct Draw Vendor Reference

For general DirectX information and the location of the DirectX SDK documentation, refer to http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx.

Appendix D - Direct3D Vendor Reference

Refer to the Direct3D Immediate Mode (d3dim) documentation contained within the DirectX SDK (http://msdn.microsoft.com/en-us/directx/default.aspx).

VIII. INDEX