Vivante Software Tool Kit

User Guide

Version 1.2

30 October 2012

This document covers the following vTools:
vAnalyzer, **vCompiler**, **vEmulator**, **vProfiler**, **vShader**, and **vTexture**.

Legal Notices

For our current distributors, sales offices, design resource centers, and product information, visit our web page located at http://www.vivantecorp.com.

Table of Contents

List of Figures

List of Tables

1 Vivante Tool Kit Overview

The Vivante Tool Kit (VTK) is a set of applications designed to be used by graphics application developers to rapidly develop and port graphics applications either stand alone, or as part of an IDE targeting a system-on-chip (SoC) platform containing an embedded GPU.

1.1 VTK Component Overview

The VTK includes a graphics and OpenCL emulator (vEmulator) to enable embedded graphics and compute application development on a PC platform, a driver and hardware performance profiling utility (vProfiler), and a visual analyzer (vAnalyzer) for graphing the performance metrics. Also provided are pre-processing utilities for stand-alone development of optimized shader programs (vShader) and for compiling shader code (vCompiler) into binary executables targeting Vivante accelerated hardware platforms. An image transfer utility (vTexture) provides compression and decompression options.

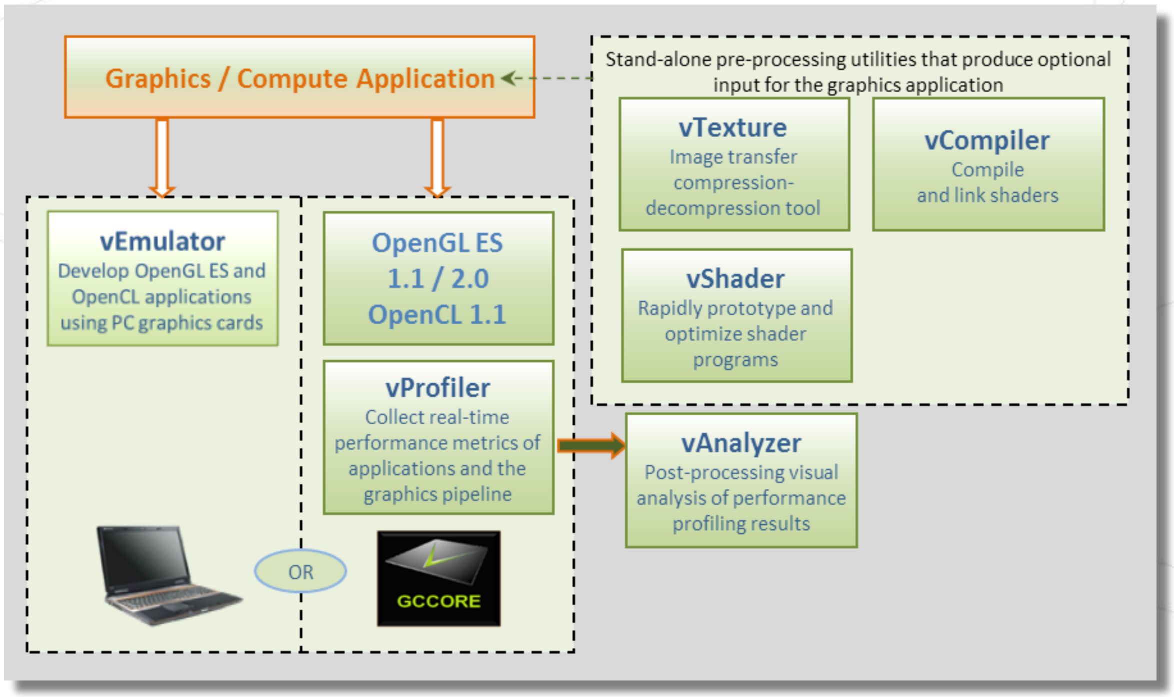

Graphics / Compute Application
Stand-alone pre-processing utilities that produce optional input for the graphics application
vTexture
Image transfer compression-decompression tool
vCompiler
Compile and link shaders
vEmulator
Develop OpenGL ES and OpenCL applications using PC graphics cards
OpenGL ES 1.1 / 2.0 OpenCL 1.1
vShader
Rapidly prototype and optimize shader programs
vProfiler
Collect real-time performance metrics of applications and the graphics pipeline
vAnalyzer
Post-processing visual analysis of performance profiling results
OR
GCCORE

Figure 1. Vivante Tool Kit vTools Components

1.2 VTK Operating System Requirements

Most VTK vTools applications are designed to run on Microsoft Windows operating systems. The following systems are compatible with current releases of vTools:

- Microsoft Windows XP Professional, with Service Pack 2 or later
- Microsoft Windows Vista with Service Pack 2 or later
- Microsoft Windows 7 Professional

Some components, such as the vProfiler, are run on other platforms. Refer to the individual vTools component detail description.

1.3 VTK Installation

The vProfiler tool is not included in the VTK. This tool can be built by setting a build command option when making the Vivante Graphics Drivers.

The VTK package contains a **`vtools`** folder. Inside this folder are five .zip packages which can be individually extracted. As an example, if you have a system with WinRAR installed, right click and select Extract Here. A folder will be created with the same name as the .zip file.

- **vAnalyzer.zip**
- **vCompiler.zip**
- **vEmulator.zip**
- **vShader.zip**
- **vTexture.zip**

Each vTools extracted folder will contain a **`SETUP.exe`** and a ***`vToolName.msi`*** file. The tool can be installed independently by running the **`SETUP.exe`** located in that tool's folder. Typical licensing and folder placement options may appear as part of the installation prompts.

vAnalyzer and vShader have a Windows GUI. vEmulator is a library. vCompiler and vTexture are utilities run from the command line.

NOTES:

- The default installation location for the VTK is usually a folder named something like **C:/Program Files/Vivante/*vToolName*,** where *vToolName* is the name of the tool being installed. Some systems may install to a Program Files (x86) folder.
- Windows navigation instructions such as Control Panel navigation will vary with the different Windows operating systems.
- Administrator rights may be required to install the tool.
- If you are installing an update version, use Windows Add/Remove programs to remove the installed version of the tool, before installing the update version.

2 vEmulator

Vivante’s vEmulator duplicates the graphics and compute functionality of the Khronos APIs—namely, OpenGL ES 2.0, OpenGL ES 1.1 and OpenCL 1.1—in a desktop PC environment. This enables developers to write and test applications for Vivante embedded GPU cores prior to their availability, using the graphics cards on Windows XP or Windows Vista™ or Windows 7 PC platforms.

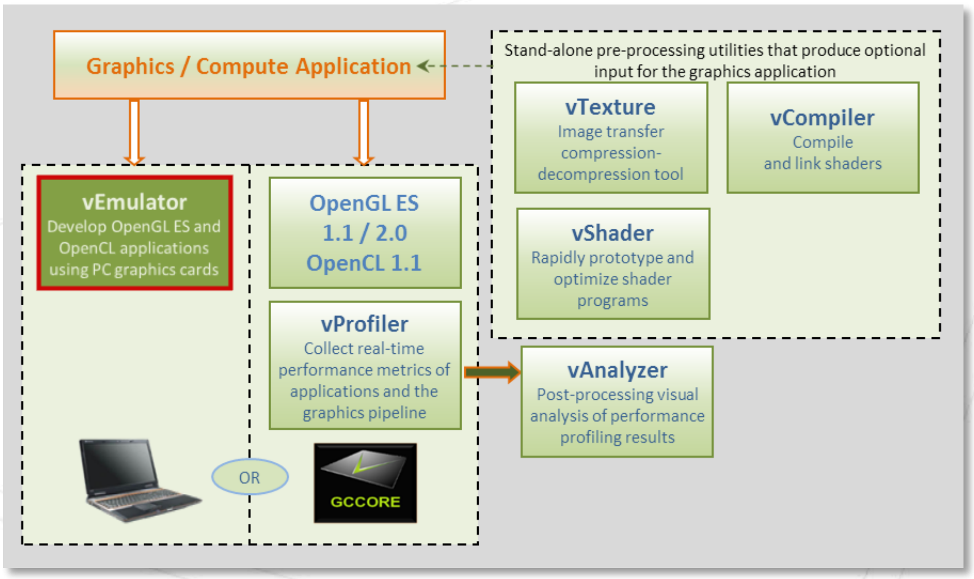

Graphics / Compute Application
Stand-alone pre-processing utilities that produce optional input for the graphics application
vTexture
Image transfer compression-decompression tool
vCompiler
Compile and link shaders
vEmulator
Develop OpenGL ES and OpenCL applications using PC graphics cards
OpenGL ES 1.1 / 2.0 OpenCL 1.1
vShader
Rapidly prototype and optimize shader programs
vProfiler
Collect real-time performance metrics of applications and the graphics pipeline
vAnalyzer
Post-processing visual analysis of performance profiling results
OR
GCCORE

Figure 2. vEmulator embedded graphics emulator

vEmulator is not an application, but rather a set of libraries that convert Khronos mobile API function calls into OpenGL 2.0 desktop or OpenCL function calls. These libraries can be accessed directly by the graphics / compute application.

2.1 Supported Operating Systems and Graphics Hardware

vEmulator libraries are available for Microsoft Windows XP, Windows Vista and Windows 7 operating systems:

- Microsoft Windows XP Professional, with Service Pack 2 or later
- Microsoft Windows Vista with Service Pack 2 or later
- Microsoft Windows 7 Professional

vEmulator has been tested on popular graphics cards, including:

- NVIDIA GeForce GTX 200 series with driver version 182.05 or later
- NVIDIA GeForce 9000 and 8000 series with driver version 182.05 or later
- NVIDIA GeForce 8400 GS with ForceWare driver version 176.44 or later
- ATI Radeon HD 3000 and 4000 series with driver version Catalyst 9.1 or later

Additional graphics cards will be added as testing is confirmed.

2.2 vEmulator Components

vEmulator libraries are packaged with the Vivante VTK installer. Once installed the libraries will reside in a folder vEmulator in the VTK installation path which can be specified by the user at time of installation. The default location of the Vivante VTK is:

```
C:\Program Files\vivante
```

The vEmulator folder contains everything that is needed for emulation. The vEmulator directory structure and its files are described in the following table.

Table 1. vEmulator Directory Contents

vEmulator subdirectory	Filename	Description
bin	libEGL.dll	Dynamic library for invoking EGL at runtime
	libGLESv1_CM.dll	Dynamic library for OpenGL ES 1.1 emulation
	libGLESv2x.dll	Dynamic library for OpenGL ES 2.0 emulation
	libOpenCL.dll	Dynamic library for OpenCL 1.1 emulation
	libVEmulatorVDK.dll	Dynamic library for vEmulator VDK functions
inc	gc_vdk.h	Vivante VDK declarations
	gc_vdk_types.h	Vivante VDK type declarations
	gc_sdk.h	Vivante SDK declarations and definitions
inc/EGL	egl.h	EGL declarations
	eglext.h	EGL extension declarations
	eglplatform.h	Platform specific EGL declarations
	eglrename.h	Rename for building static link driver
	eglunname.h	For mixed usage of ES11, ES20
	eglvivante.h	Vivante EGL declarations
inc/GLES	egl.h	EGL declarations
	gl.h	OpenGL 1.1 declarations
	glext.h	OpenGL1.1 extension declarations
	glplatform.h	Platform specific OpenGL 1.1 declarations
	glrename.h	Rename for building static link driver
	glunname.h	For mixed usage of ES11, ES20
inc/GLES2	gl2.h	OpenGL 2.0 declarations
	gl2ext.h	OpenGL 2.0 extension declarations
	gl2platform.h	Platform specific OpenGL 2.0 declarations
	gl2rename.h	Rename for building static link driver
	gl2unname.h	Unified name definitions
inc/hal	gc_hal_eglplatform_type.h	Vivante HAL platform specific struct declarations
inc/KHR	khrplatform.h	Platform specific Khronos declarations
lib	libEGL.lib	Static library for linking EGL functions
	libGLESv1_CM.lib	Static library for linking OpenGL ES 1.1 functions
	libGLESv2x.lib	Static library for linking OpenGL ES 2.0 functions
	libVEmulatorVDK.lib	Static library for linking vEmulator VDK functions
samples/es11, /es20	tutorials.sln	Microsoft Visual Studio project solution file for samples
samples/es11/tutorial*N*	-- Varies with *N* --	Sample OpenGL ES 1.1 applications
samples/es20/tutorial*N*	-- Varies with *N* --	Sample OpenGL ES 2.0 applications

2.3 vEmulator for OpenCL

If your edition of vEmulator includes support for OpenCL, additional files may be present. For OpenCL emulation using vEmulator on your PC, please refer to the OpenCL emulator readme file (OCL_Readme.txt) in the vEmulator folder for additional installation instruction.

Note: An additional environment variable **CL_ON_GC2100** needs to be set for simulation for GC2100. The value can be any characters, as long as it is not null. This variable does not need to be set for other OCL cores.

Table 2. vEmulator Files for OpenCL 1.1

vEmulator subdirectory	Filename	Description
	OCL_Readme.txt	Readme file for OpenCL 1.1
bin	libOpenCL.dll	Dynamic library for invoking OCL at runtime
inc/CL	cl.h	OpenCL 1.1 core API header file
	cl.hpp	OpenCL 1.1 C++ binding header file
	cl_d3d10.h	OpenCL 1.1 Khronos OCL/Direct3D extensions header file
	cl_ext.h	OpenCL 1.1 extensions header file
	cl_gl.h	OpenCL 1.1 Khronos OCL/OpenGL extensions header file
	cl_gl_ext.h	OpenCL 1.1 Vivante OCL/OpenGL extensions header file
	cl_platform.h	Platform specific OCL declarations
	opencl.h	Vivante HAL version
lib	libOpenCL.lib	Dynamic library for linking OpenCL functions
samples/cl11	cl_sample.cpp	Sample OpenCL 1.1 source code
samples/cl11	cl_sample.sln	Sample OpenCL 1.1 Visual Studio solution file
samples/cl11	cl_sample.vcproj	Sample OpenCL 1.1 Visual Studio solution project file
samples/cl11	square.cl	Sample OpenCL 1.1 kernel file

2.4 Supported Extensions

Refer to the document **EGL & OES Extensions Support** for a list of supported and custom extensions available for EGL and OpenGL ES.

Software extensions have not been added to vEmulator for OpenGL ES 2.0. vEmulator relies on the extensions available with the installed version of native OpenGL.

2.5 vEmulator Environment Variable Setup

There are two steps to running an OpenGL ES or OpenCL application with vEmulator:

Step 1. Link to the vEmulator *.lib static libraries at build time when creating an application executable image.

Step 2. Provide a path to the vEmulator *.dll dynamic libraries during run-time.

These steps require a one-time setup in which the location of the vEmulator libraries is added to the Microsoft Windows system environment variable named "Path." In our example, the following string would be added to the system "Path" variable: `C:\Program Files\vivante\vEmulator\lib.`

To add vEmulator DLL files to the Windows XP system path:

a. Click **Start** then click **Control Panel** then double-click **System**
   - Vista: then click **Advanced system settings** from the Tasks list in the upper-left corner of the window.
   - Windows 7: in the System and Security window, click System, then on the left menu column click **Advanced system settings**.

b. Select the **Advanced** tab, then click on the **Environment Variables…** button.
   - An Environment Variables dialogue box will pop up, with two panes for variables.

c. Select **Path**, and then click on the **Edit…** button.

d. In the **Variable value:** field type the following environment variables in the order you want them found. For instance:

   C:\Program Files\vivante\vEmulator\lib; <current path>

   Note: The system parses a path string in left-to-right order when looking for a file. Whatever it finds first is what will be used.

e. If the Vivante Core is GC2100, an additional variable **CL_ON_GC2100** should be set to any non-null value.

f. Click **OK**.
   - Click **OK** to close the Environment Variables dialogue window.
   - Click **OK** to close the System Properties dialogue window.
   - Close the Control Panel > System window.

2.6 Sample Code Overview

In the discussions that follow about the various sample programs included with the vEmulator distribution, we assume that vEmulator has been installed in the default location within the vivante VTK folder:

`C:\Program Files\vivante\vEmulator`.

Relative to this path:

• run-time dlls are located at	**…\bin**
• include-files are found at	**…\inc**
• library files are located at	**…\lib\<API>**
• examples are located at	**…\samples\<API>\tutorial*** where API is one of: **es11** or **es20**

The code examples are distributed with working *.exe executable images so that the VTK user can see how the results should look.

They are presented in a tutorial fashion, progressing from simpler programs to more complex as the tutorial number increases.

2.7 Building and Running the Code Examples

The steps to build and run are identical for all code examples, regardless of the API (es11 or es20). There are two general guidelines to keep in mind.

1) A Visual Studio project has environment variables that allow the specification of additional paths to “include” and “library” files when a source module from that project is being built. The Visual Studio projects that are part of the vEmulator distribution package are configured out-of-the-box for building all of the sample code executables, relative to the location where vEmulator is installed. Specifically the additional paths are set as “$(SolutionDir)..\..\inc” and “$(SolutionDir)..\..\lib”.

   If \samples is moved, or if the VTK user begins with the provided projects as templates for developing applications in a directory that is not directly under the \vEmulator installation, then the project path variables must be adjusted accordingly. For example:

   To access these path variables for tutorial1, first launch the tutorials.sln

   - Right-click on **tutorial1**, then select **Properties** (at the bottom of the pop-up menu)

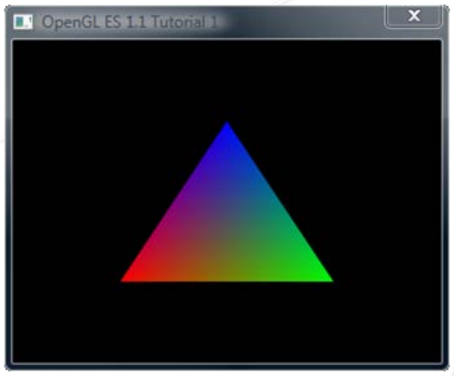
   - Under “Configuration Properties” > “C/C++” > “General”, edit the **Additional Include Directories** entry
     - E.g., change **`..\..\..\inc`** to **`C:\Program Files\vivante\vEmulator\inc`**

   - Under “Configuration Properties” > “Linker” > “General”, edit the **`Additional Library Directories`** entry
     - E.g., change **..\..\..\lib** to **`C:\Program Files\vivante\vEmulator\lib`**

2) Make sure that the system environment variable**PATH** contains a path to the vEmulator DLL files. (See above section on vEmulator Environment Variable Setup, above.) Remember that the path is order-dependent; whatever the system finds first will be used. If there is more than one DLL with the same name, double-check that the path to the desired one is listed first in the **PATH** string.

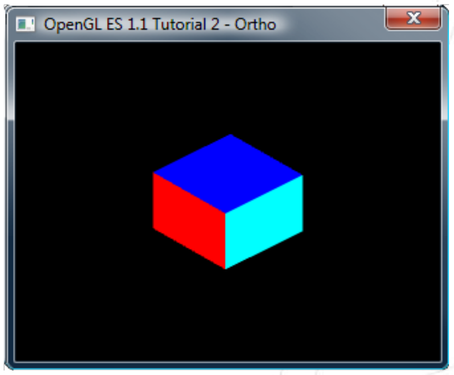
2.8 OpenGL ES 1.1 Examples

2.8.1 tutorial1: Rotating Three Color Triangle



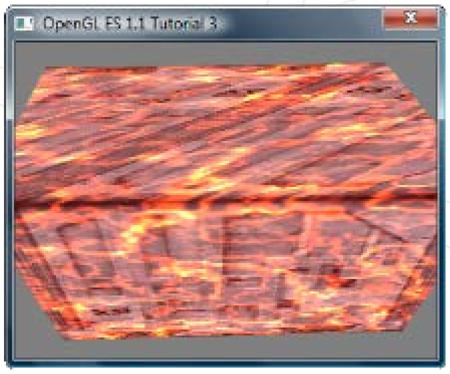
Renders a cube centered at the origin with a different color on each face. Flat shading is used. The cube rotates about the vertical axis. The default projection is ORTHO, which can be toggled between ORTHO and PERSPECTIVE by left-clicking in the display window with the mouse or pressing Enter.

2.8.2 tutorial2: Rotating Six-color Cube



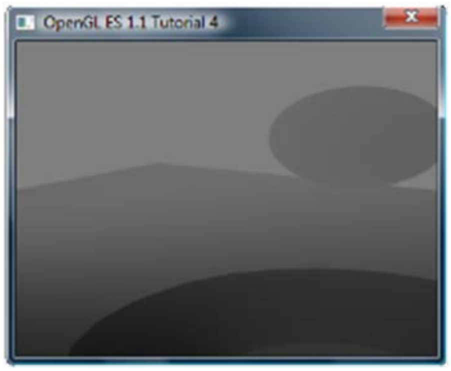
Renders a cube centered at the origin with a different color on each face. Flat shading is used. The cube rotates about the vertical axis. The default projection is ORTHO, which can be toggled between ORTHO and PERSPECTIVE by left-clicking in the display window with the mouse or pressing Enter.

2.8.3 tutorial3: Rotating Multi-Textured Cube



This example takes the cube of the previous example with PERSPECTIVE projection, loads two textures from file and combines them using GL_ADD blending mode, and applies the resulting texture to the cube faces.

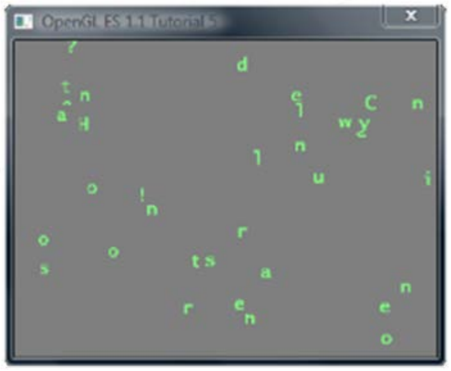
2.8.4 tutorial4: Lighting and Fog



What appears to be a torus, a cone, and an oblate spheroid orbiting about the center of a plane is actually a single mesh being lit by a single rotating, diffuse light source. Green fog is added to the scene by left-clicking on the display window with the mouse or pressing Enter.

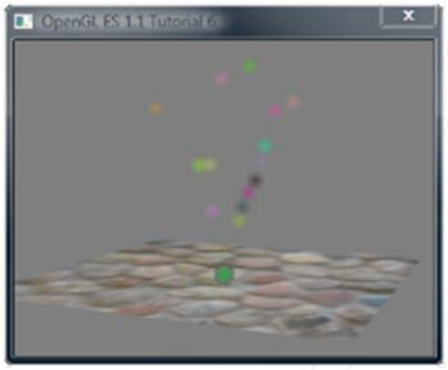
2.8.5 tutorial5: Blending and Bit-mapped Fonts

This example makes use of alpha blending to animate sprites across the display, and it also instructs how to create a bit-mapped font from a texture. Jumbled letters iteratively print and move across the display as they unscramble into a text message.



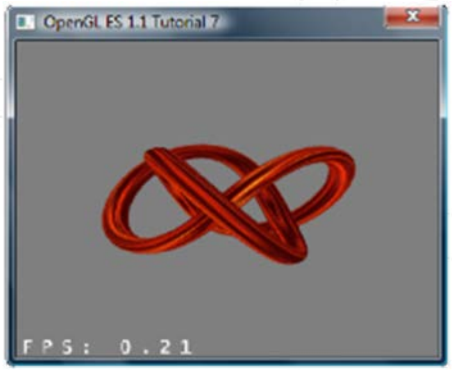
2.8.6 tutorial6: Particles Using Point Sprites

This example reuses the bit-mapped font technique from the previous tutorial, but it adds a particle generator to simulate and animate particles being emitted from the textured plane. All computation is performed in fixed-point arithmetic.



2.8.7 tutorial7: Vertex Buffer Objects

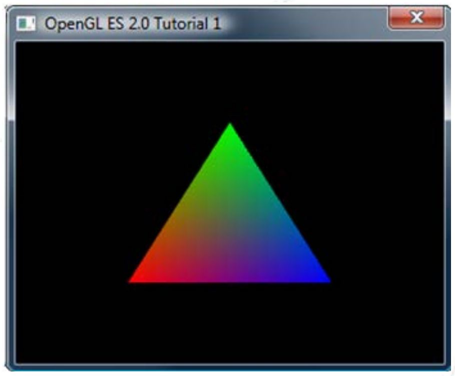
Using Vertex Buffer Objects (VBO) can substantially increase performance by reducing the bandwidth required to transmit geometry data. Information such vertex, normal vector, color, and so on is sent once to locate device video memory and then bound and used as needed, rather than being read from system memory every time. This example illustrates how to create and use vertex buffer objects.



2.9 OpenGL ES 2.0 Examples

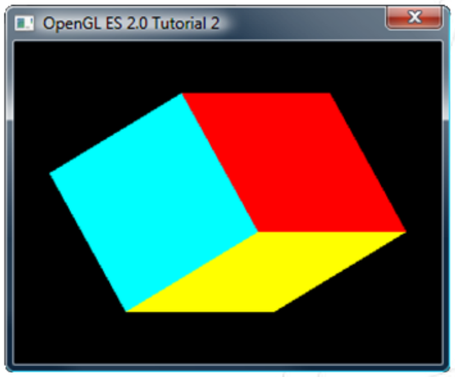
2.9.1 tutorial1: Rotating Three-color Triangle

A single triangle is rendered with a different color at each vertex, Gouraud shading for blending, rotational animation in the final display. This is the same example as es11/tutorial1, only implemented in OpenGL ES 2.0.



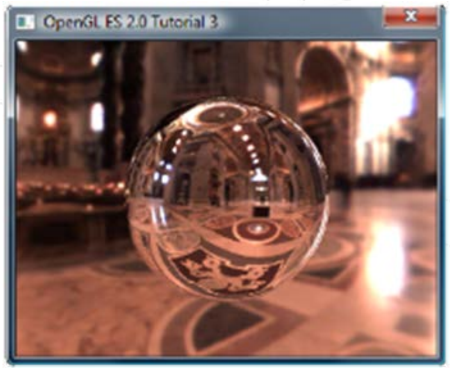
2.9.2 tutorial2: Rotating Six-color Cube

Renders a cube centered at the origin with a different color on each face, and rotates it about the vertical axis. Similar to the es11/tutorial2 example, the default projection is ORTHO. But there is no toggle for PERSPECTIVE.



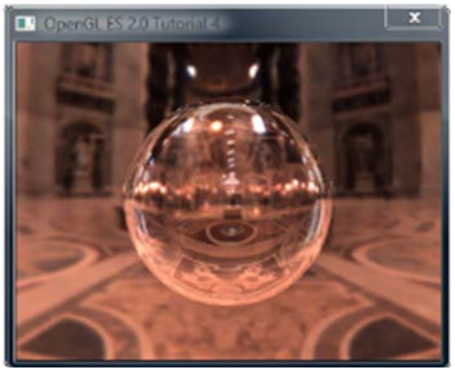
2.9.3 tutorial3: Rotating Reflecting Ball

A ball made of a mirroring material and centered at the origin spins about its Y-axis and reflects the scene surrounding it.



2.9.4 tutorial4: Rotating Refracting Ball

This example is the same as the previous one, except that the ball is made of clear glass which refracts the surrounding environment.



3 vShader

vShader is a complete off-line environment for editing, previewing, analyzing, and optimizing shader programs.

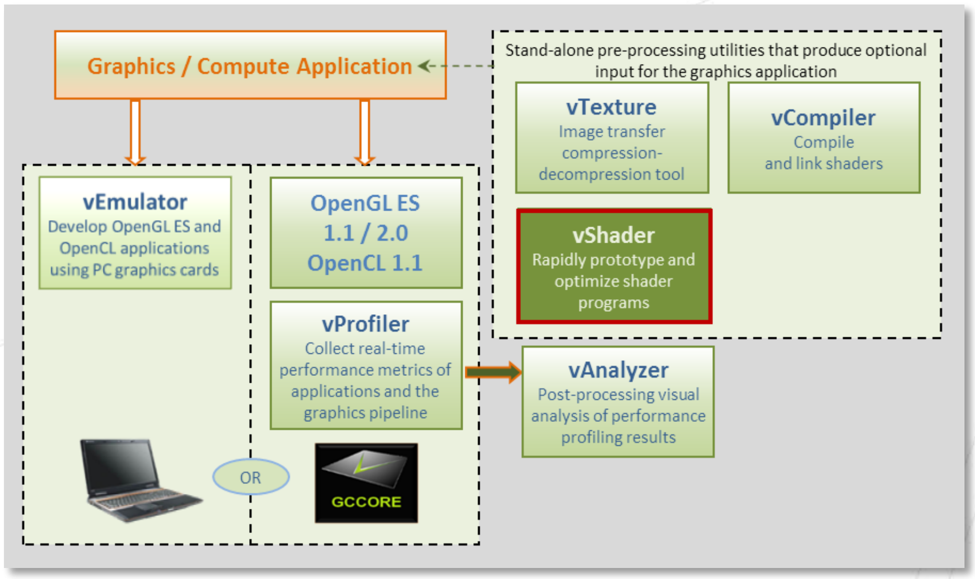

Graphics / Compute Application
Stand-alone pre-processing utilities that produce optional input for the graphics application
vTexture
Image transfer compression-decompression tool
vCompiler
Compile and link shaders
vShader
Rapidly prototype and optimize shader programs
vEmulator
Develop OpenGL ES and OpenCL applications using PC graphics cards
OpenGL ES 1.1 / 2.0 OpenCL 1.1
vProfiler
Collect real-time performance metrics of applications and the graphics pipeline
vAnalyzer
Post-processing visual analysis of performance profiling results
OR
GCCORE

Figure 3. vShader shader editor

vShader allows users to:

- Map any texture onto shaders
- Import user-defined meshes
- Bind mesh attributes to shaders
- Set uniforms in shaders
- View shader compiler output for optimization hints
- Predict hardware performance

3.1 vShader Components

By default, the vShader executable installs in the following location within the Vivante Toolkit directories: **`C:\Program Files\vivante\vShader`**

The vShader package includes samples of shader programs, a number of standard meshes (sphere, cube, tea pot, pyramid, etc.) and a text editor. These extra features will help programmers get a quick start on creating their shader programs.

By combining vertex shaders and fragment shaders into a single shader program, an application can produce a shader effect. A project can make use of many shader effects, which can share vertex and fragment shaders, mixing and matching to achieve the desired results.

The scope of this guide is to cover the vShader user interface. The tutorials provided with the vShader package are there to help the reader learn about shaders, if needed.

3.2 Getting Started with vShader

Once the vShader utility is launched by clicking on a shortcut or directly on the executable vShader.exe projects can be created, developed and saved. Project files have an extension **.vsp**.

3.2.1 Creating a new project

To create a new project, locate the main menu bar: Select **File** then **New Project…**

Depending on the current project status, one of three things will happen:

1. If this is the first time vShader is launched, then there is no project already open and selecting "File > New Project…" will appear to have no effect.
2. If there have been no changes to the current project since the last save, then the current project will close and a new, empty project will be opened.
3. If the current project has been modified, then a dialog box will pop up to ask if you want to save the changes. Choosing **Yes** will commit the changes to the current project, which will then be closed, and a new, empty project will be opened.

3.2.2 Opening an existing project

To open an existing project, locate the main menu bar:

1. Select **File** then **Open Project…**
2. Double-click on the desired project from the list that pops up, or single-click on the project name and click **OK**.

The project will load into vShader and appear in the state it was last saved.

3.2.3 Saving a project

To save a project, locate the main menu bar:

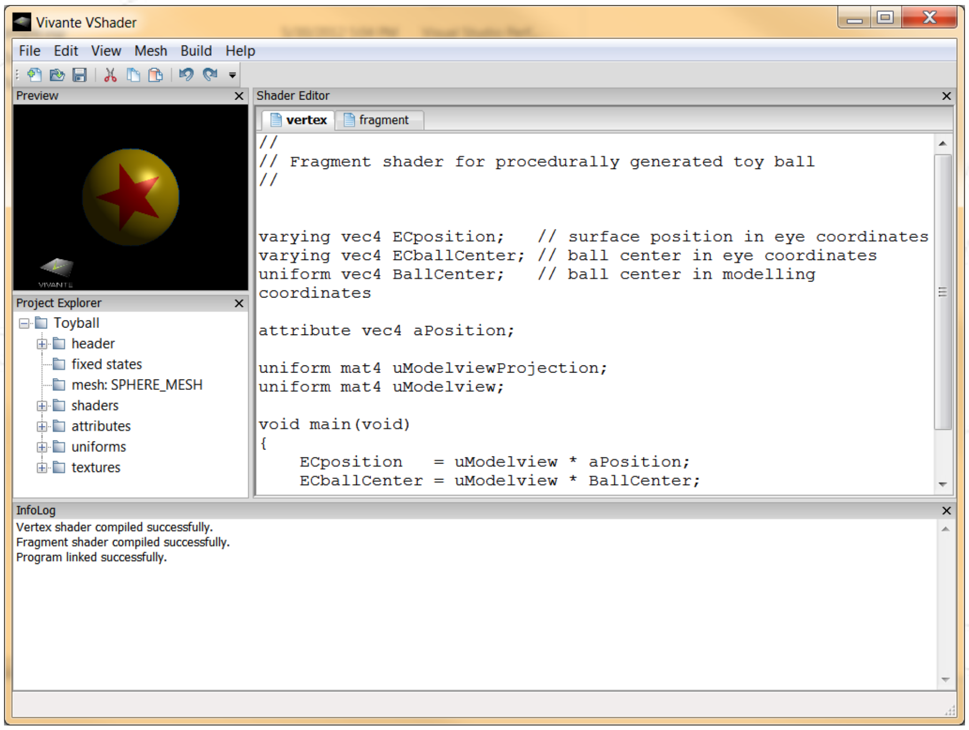
1. Select **File** then **Save Project…**
2. In the resulting dialog box indicate where to save the project, then click **OK.**

3.3 vShader Navigation

The vShader application runs on the Windows XP, Windows Vista and Windows 7 platforms and is driven from a graphical user interface as shown in the figure below.

Main components of the GUI include:

- on upper portion of window: a Menu Bar, Menu Icons,
- on left: Preview pane, Project Explorer pane
- on right: Shader Editor pane, and
- on lower portion of window: InfoLog pane.



```
//
// Fragment shader for procedurally generated toy ball
//


varying vec4 ECposition;   // surface position in eye coordinates
varying vec4 ECballCenter; // ball center in eye coordinates
uniform vec4 BallCenter;   // ball center in modelling
coordinates

attribute vec4 aPosition;

uniform mat4 uModelviewProjection;
uniform mat4 uModelview;

void main(void)
{
    ECposition   = uModelview * aPosition;
    ECballCenter = uModelview * BallCenter;
```

```
Vertex shader compiled successfully.
Fragment shader compiled successfully.
Program linked successfully.
```

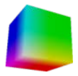
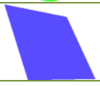
Figure 4. vShader GUI main window

3.3.1 vShader Menu Bar

The main window opens when a user launches vShader. The main menu bar contains drop-down menus for File, Edit, View, Mesh, Build, and Help.

Table 3. vShader Menu Commands

Menu Name	Menu Command	Description
File		
	New Project…	Create a new project file; if a project is currently open, then the user is prompted to choose whether to save it first.
	Open Project…	Browse for and load a .vsp VShader project.
	Save Project…	Save the current project; if this is the first time saving this project, then the user is prompted to choose where to save it.
	Load Vertex…	Browse for and load a vertex shader from an existing text file.
	Load Fragment…	Browse for and load a fragment shader from an existing text file.
	Save Vertex Shader As…	Prompts for filename and location to save the active vertex shader.
	Save Fragment Shader As…	Prompts for filename and location to save the active fragment shader.
	Exit	Close all open files and exit VShader.
Edit		
	Undo [Ctrl-z]	Revert to a previous edit state (Note: Undo is only 1-level deep)
	Redo [Ctrl-z]	Re-apply the last "undone" edit command (Note: Redo is only 1-level deep)
	Cut [Ctrl-x]	Delete the selected item(s) and save a copy in the paste buffer
	Copy [Ctrl-c]	Save a copy of the selected item(s) item in the paste buffer
	Paste [Ctrl-v]	Insert the contents of the paste buffer
	Delete [Del or Bkspc]	Remove the selected item(s)
	Select All [Ctrl-a]	Highlight all items in the current view
View		
	Reset Preview	Reset Preview window.
	Snapshot	Save current preview image to bitmap bmp file. A dialog will display to let user choose where to save the bmp.
	Perspective	Use perspective projection in the Shader Preview pane
	Ortho	Use orthographic projection in the Shader Preview pane
	Tool Bar	Show or hide toolbar icons
	Preview Window	Show or hide Preview window
	Project Explorer	Show or hide Project Explorer window
	Shader Editor	Show or hide Shader Editor window
	InfoLog	Show or hide InfoLog window

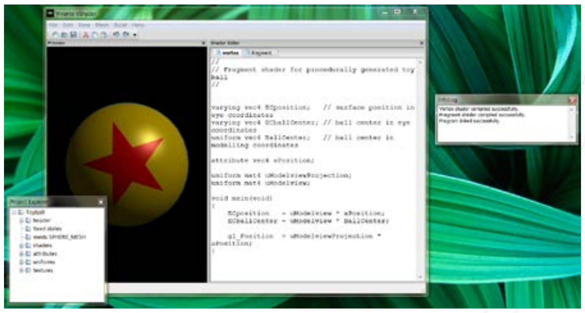
Mesh			
	Conic	Looks like a spiral horn.	
	Cube	A 3D cube.	
	Klein	The Klein bottle.	
	Plane	A 2D square.	
	Sphere	A ball.	
	Teapot	The Utah teapot.	
	Torus	Looks like a donut.	
	Trefoil	A trefoil knot.	
	Custom Mesh…	Browse for and open a 3DS mesh file.	
Build			
	Compile	Compile the active shader; this command is also available via the "Compile" button in the Shader Editor window pane.	
	Link	Link the vertex and fragment shaders into a shader program, and apply it to the mesh showing in the Shader Preview window pane.	
	Clear InfoLog	Remove all text currently showing in the InfoLog window pane.	
Help			
	About	Information about the version of VShader being used.	

3.3.2 vShader Window Panes

There are four window panes in the vShader GUI: Preview, Project Explorer, Shader Editor, and InfoLog. Each pane can be resized by left-mouse-dragging the pane edge. A pane can be hidden by clicking the **X** in the upper-right corner of the pane, or by un-checking the box next to its name in the View pull-down of the main menu. Restoring a hidden window pane is done by checking the appropriate box in the View pull-down menu.

Individual panes in the vShader application can be resized, relocated or converted to detached windows, as in the example to the right.

Note: Changes made to pane arrangement are not restored on application or project relaunch.



```
//
// Fragment shader for procedurally generated toy
ball
//

varying vec4 ECposition;   // surface position in
eye coordinates
varying vec4 ECballCenter; // ball center in eye
coordinates
uniform vec4 BallCenter;   // ball center in
modelling coordinates

attribute vec4 aPosition;

uniform mat4 uModelviewProjection;
uniform mat4 uModelview;

void main(void)
{
    ECposition   = uModelview * aPosition;
    ECballCenter = uModelview * BallCenter;

    gl_Position  = uModelviewProjection *
aPosition;
}
```

```
Vertex shader compiled successfully.
Fragment shader compiled successfully.
Program linked successfully.
```

Figure 5. vShader Moveable Panes

3.3.2.1 Preview

The shader Preview pane shows the current effect of the shaders on the chosen mesh geometry. A different mesh may be chosen either via the Mesh pull-down menu in the menu bar near the top of the vShader main window or by right-mouse clicking in the Preview pane.

When using the right-click method, the user also can choose between perspective and orthographic views of the mesh, can reset the view orientation to the default, or can save the current view in the Preview window as a bitmap file by selecting **Snapshot**.

The object in the Preview window can be rotated, translated, and scaled. Rotation is controlled by left-mouse-drag; translation is done by holding the Ctrl key plus left-mouse-drag; scaling the image is seen by holding the Alt key while applying left-mouse-drag.

When shader variables are changed, the shader preview updates automatically. When shader programs are changed they must be recompiled and relinked by the user, through the Build menu. The Preview display will automatically update to reflect the new Build.

3.3.2.2 Project Explorer

The Project Explorer displays all of the project resources in a familiar tree structure. The root of the tree is the project name, and the branches and leaves classify the resources. Folders can be expanded by clicking on the plus sign next to them, and they can be collapsed by choosing the minus sign. By right-mouse clicking on any resource name, the user can view and usually edit that resource.

3.3.2.3 Shader Editor

The Shader Editor is a work area for entering and modifying shader programs. There are two tabs: one for vertex shader, and one for fragment shader. Changes made to a shader must be compiled and linked in order for their effect to appear in the Shader Preview.

Compiling can be done by selecting **Build** then **Compile** from the main menu bar. Likewise, linking and applying the shaders is performed by choosing **Build** then **Link**.

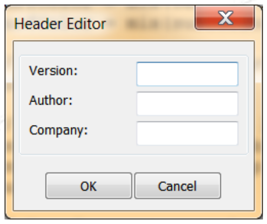
3.3.2.4 Info Log

The Info Log window pane receives diagnostic messages from the compiler and linker, so that the user can see if the current shaders have built without errors. This pane can be cleared of text by selecting the **Build** then **Clear InfoLog** entry in the main menu.

3.4 vShader Project Resources

Project resources are accessible from the Project Explorer pane. Click on the item and an Editor pop-up dialog will appear where the user can enter alternate values. Resources include: header, fixed states, mesh, shaders, attributes, uniforms, and textures.

3.4.1 Header

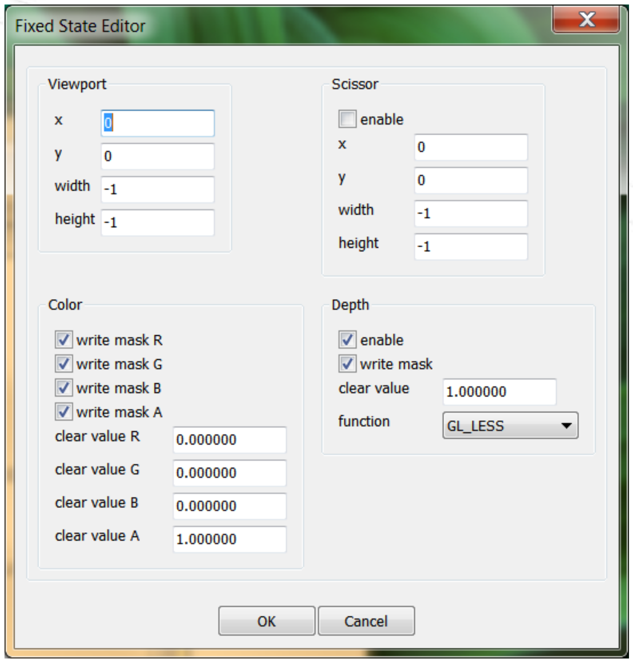


Header Editor

Version:	
Author:	
Company:	

Some project identifying information, namely version, author, and company. Expand the folder to see the settings, or right-click (or double-click) the folder to edit them.

3.4.2 Fixed States



Fixed State Editor

Viewport

x	0
y	0
width	-1
height	-1

Scissor

☐ enable

x	0
y	0
width	-1
height	-1

Color

☑ write mask R
☑ write mask G
☑ write mask B
☑ write mask A

clear value R	0.000000
clear value G	0.000000
clear value B	0.000000
clear value A	1.000000

Depth

☑ enable
☑ write mask

clear value	1.000000
function	GL_LESS

The Fixed State Editor is a list of OpenGL ES 2.0 fixed states settings, such as depth test enable/disable, etc. It allows the user to set all fixed states manually. Right-click or double click to display an edit dialog.

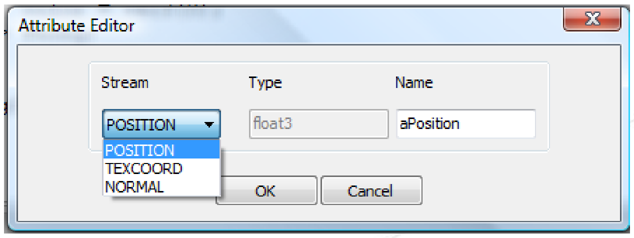
3.4.3 Mesh

This resource shows the name of the mesh which is currently being displayed in the Preview pane. It does not have a pop-up window. Right-click on the mesh name to select a different mesh can be selected from the resulting pull-down menu.

3.4.4 Shaders

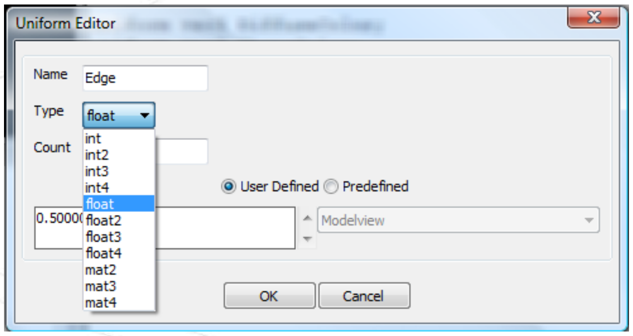
Left-click on the plus sign next to the “shaders” folder to reveal the two sub nodes in this section, which are vertex and fragment. Double-click (or right-click and then choose **Active**) on either shader to bring it forward in the Shader Editor for editing.

3.4.5 Attributes



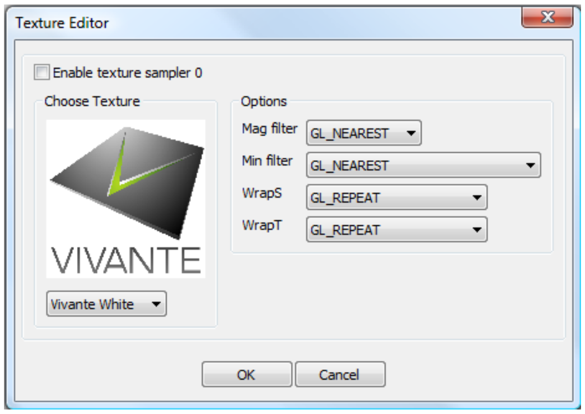
The Attribute Editor dialog displays all attributes bound to the current project. It allows the user to add new attributes, and edit or remove existing attributes. Right-click on **Attributes** to add a new one. Click on the plus sign to expand the attributes list, and then double-click to edit a particular attribute. Also, by right-clicking on an attribute, you can edit or remove that attribute or add a new one. Up to 12 attributes are allowed.

3.4.6 Uniforms



This displays all uniforms bound to the current project. Right-click on **Uniforms** to add a new one, or expand the list and double-click on a given uniform to bring up the Uniform Editor dialog. When a uniform is right-clicked, the user can add new uniforms, or edit or remove existing uniforms. Up to 160 uniforms are allowed.

3.4.7 Textures



The Texture Editor dialog allows the user to select a texture for each of up to 8 texture units. The effect of applying each texture is seen immediately in the Shader Preview pane.

The texture selection option list is created from the texture files located in the "textures" subfolder of the project. The list can be expanded by adding textures to the textures folder, formatted as bitmap files.

4 vCompiler

vCompiler is an off-line compiler and linker for translating vertex and fragment shaders written in OpenGL ES Shading Language (ESSL) into binary executables targeting Vivante accelerated hardware platforms. vCompiler is driven by a simple command-line interface.

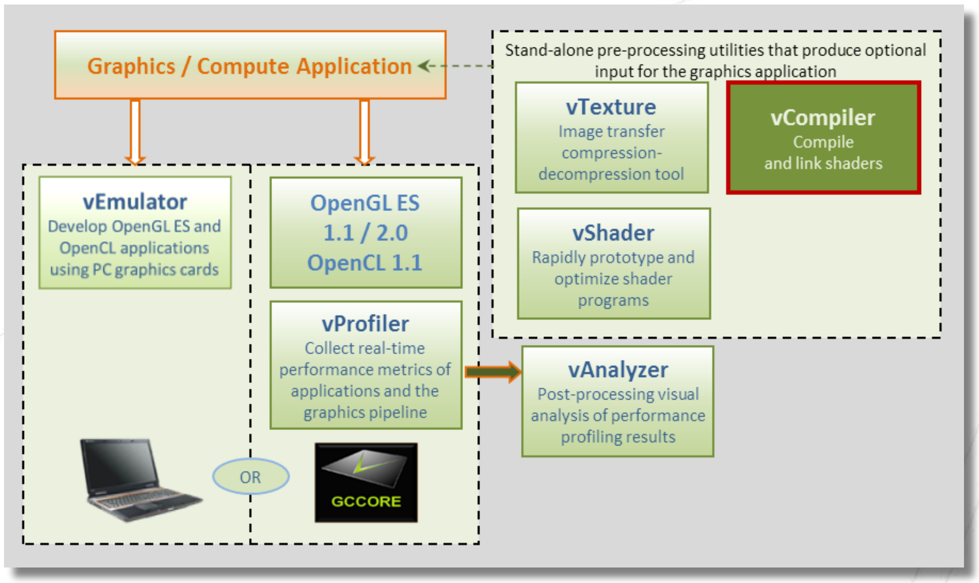

Graphics / Compute Application
Stand-alone pre-processing utilities that produce optional input for the graphics application
vTexture
Image transfer compression-decompression tool
vCompiler
Compile and link shaders
vEmulator
Develop OpenGL ES and OpenCL applications using PC graphics cards
OpenGL ES 1.1 / 2.0 OpenCL 1.1
vShader
Rapidly prototype and optimize shader programs
vProfiler
Collect real-time performance metrics of applications and the graphics pipeline
vAnalyzer
Post-processing visual analysis of performance profiling results
OR
GCCORE

Figure 6. vCompiler compiler/linker

4.1 vCompiler Command Line Syntax

4.1.1 Syntax:

Optional inputs are indicated by italic font.

```
vCompiler <shaderInputFileName> [shaderInputFileName_2] [ -c ] [ -h ] [ -l ]
       [ -o <outputFileName> ] [ -On ] [ -v ] [ -x <shaderType> ]
```

4.1.2 Input parameters (required):

shaderInoutFileName shader input file name, which **must** contain one of the following file extensions:

vert	vertex shader source file
frag	fragment shader source file
vgcSL	previously compiled vertex shader input/output file
pgcSL	previously compiled pixel shader input/output file

4.1.3 Input parameters (optional):

shaderInputFileName_2 up to two shader files can be specified. The second shader file is optional but must have one of the file extensions described above for shaderInputFileName. If the first shader is a vertex shader, this second shader should be a fragment shader; conversely if the first shader is a fragment shader, the second should be a pixel shader.

	Note: pre-compiledand compiled shaders may be mixed, as long as one is a vertex shader and the other a fragment shader.
-c	Compile each vertex .vert file into a **vgcSL** file and/or fragment shader .frag file into a **pgcSL** only, with no merged result file of type **.gcPGM**. If the **–c** option is not specified: a) When only one shader is specified, that shader will be compiled into a .[v/p]gcSL file. b) When two shaders are specified, one is assumed to be a vertex shader and the other a fragment shader. Each shader can be either a previously compiled .vgcSL or .pgcSL. file or a.vert or .frag still to be compiled. The two will be merged into a .gcPGM file after successful compilation.
-h	Shows a help message on all the command options.
-l	Create a log file. The log file name is created by taking the first input file name, then replacing its file extension with ".log". If the input file name does not have a file extension, .log is appended. e.g., **myvert.vert** => **myvert.log** **inputfrag** => **inputfrag.log**
-o <outputFileName>	Specify the output file name. If the path is other than the current directory, it must also be specified. Any extension can be specified. If the extension is not specified, the following are **outputFileName** supported default types: vgcSL compiled vertex shader output file, usually compiled from a .vert input source file (default result for single file compile) pgcSL compiled pixel shader output file, usually compiled from a .frag source input file. gcPGM compiled file merging vertex shader and fragment/pixel shader into a single output file
-O*n*	Optimization level. Default is **–O1:** **-O0** Disable optimizations **-O1- -O9** Indicates on which level optimization should be done. The default is level 1. Note: Optimization is actually implemented in the compiler, not vCompiler.
-v	Verbose; prints compiler version and diagnostic messages to STDOUT.
-x <shaderType>	Explicitly specifies the type of shader instead of relying on the file extension. This option applies to all following input files until the next **-x** option. **ShaderType**: supported values for Shader type include: **vert** vertex shader source file **frag** fragment shader source file **vgcSL** compiled vertex shader input/output file **pgcSL** compiled pixel shader input/output file
-x none	revert back to recognizing shader type according to the file name extension.

4.1.4 vCompiler Output

Output files are placed in the current directory, unless another directory is specified with the –o option. The files can be of the three types described above under outputFileName value of the –o option.

4.1.5 vCompiler Syntax Examples

`vCompiler foo.vert`	produces foo.gcSH
`vCompiler bar.frag`	produces bar.gcSH
`vCompiler foo.vert bar.frag`	produces foo.gcPGM
`vCompiler –v –l –O1 foo.vert bar.frag`	produces foo.gcPGM and foo.log
`vCompiler –v –l –O1 –o foo_bar foo.vert bar.frag`	produces foo_bar.gcPGM and foo_bar.log

5 vTexture

The Vivante vTexture tool is introduced in software release version 4.6.9. vTexture is a command line tool which provides compression and decompression functions to help developers transfer image formats.

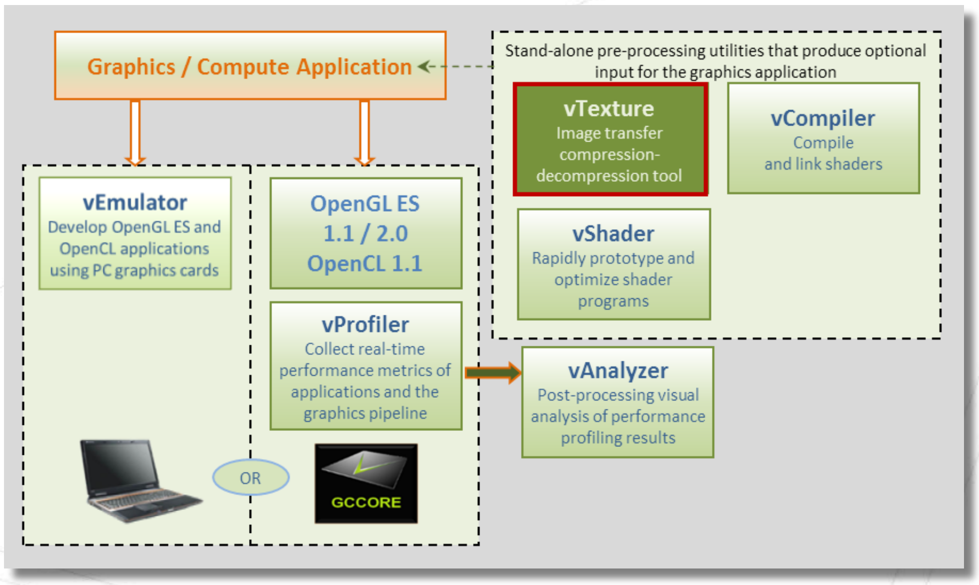

Graphics / Compute Application
Stand-alone pre-processing utilities that produce optional input for the graphics application
vTexture
Image transfer compression-decompression tool
vCompiler
Compile and link shaders
vShader
Rapidly prototype and optimize shader programs
vEmulator
Develop OpenGL ES and OpenCL applications using PC graphics cards
OpenGL ES 1.1 / 2.0 OpenCL 1.1
vProfiler
Collect real-time performance metrics of applications and the graphics pipeline
vAnalyzer
Post-processing visual analysis of performance profiling results
OR
GCCORE

Figure 7. vTexture Image Transfer Tool

5.1 Formats

5.1.1 Supported Formats

The vTexture tool supports:

- compression of uncompressed TGA format files to any of the following formats:
  - DXT1
  - DXT3
  - DXT5
  - ETC1
- decompression to uncompressed TGA format of the following compressed format file types:
  - DXT1
  - DXT3
  - DXT5
  - ETC1

The compressed DXTn format image file will be stored as a DDS file, and the ETC1 format image will be stored as a PKM file.

5.1.2 Format Limitations

The TGA format uses the RGBA color model and ETC1 format provides an image following the RGB color model. Therefore, compressing a TGA image to ETC1 format will result in a loss of alpha values.

5.2 Command Line Syntax

Open a Command prompt.

Navigate to the folder which contains the vTexture files (for example, **C:/Program Files (x86)/vivante/vTexture)**.

Launch the **vTexture** or **vTextureTools** application using the command line syntax described below.

5.2.1 Syntax

The usage of the command line tool is as follows:

```
vTextureTools -c [option] -src [infile] –dest [outfile]
```

or

```
vTextureTools -d [option] -src [infile] –dest [outfile]
```

5.2.2 Parameters

-c compress a source image of format uncompressed TGA

[option] specify the target output compression format:

-DXT1	compress image to DXT1 format (default format).
-DXT3	compress image to DXT3 format.
-DXT5	compress image to DXT5 format.
-ETC1	compress image to ETC1 format

-d decompress a source image of format specified by the value specified by [option]. The resulting filetype will be uncompressed TGA.

[option] The decompress options are is:

-TGA	decompress DXT1, DXT3, DXT5 or ECT1 format image to TGA format.

-src [infile] source file - input image path and filename.
Note: For option **–c** compress, the application expects an input filename with a .TGA extension; for **–d** decompression the application expects .DDS or .PKM extension.

-dest [outfile] destination file - image path and filename.
Note: the application expects a filename with a .TGA, .DDS or .PKM extension.

-h show help

5.2.3 vTexture Output

Output from the compress option:

- DXTn format image file will be stored as a DDS file,
- ETC1 format image will be stored as a PKM file.

Output from the decompress option:

- all supported formats will be decompressed to an uncompressed TGA file.

5.2.4 vTexture Syntax Examples

```
vTextureTools -c tga -src C:/vtexinmyfile.tga –dest C:/vtexout/myfile.dds
vTextureTools -d etc1 -src C:/vtexin/myfile2.pkm –dest C:/vtextout/myfile2.tga
vTextureTools -d -src C:/vtexin/myfile3.dds –dest C:/vtextout/myfile3.tga (assumes DXT1)
```

6 vProfiler and vAnalyzer

vProfiler is a run-time environment for collecting performance statistics of an application and the graphics pipeline. vAnalyzer is a utility for graphically displaying the data gathered by vProfiler and aiding in visual analysis of graphics performance. Used together, these tools can assist software developers in optimizing application performance on Vivante enabled platforms. The GPU includes performance counters that track a variety of GPU functions. vProfiler gathers data from these counters during runtime and can track data for a range of frames or a single frame from any application. Appendix A contains a partial list of the data gathered by the hardware performance counters. Additional counters are present in the software drivers and hardware access layer.

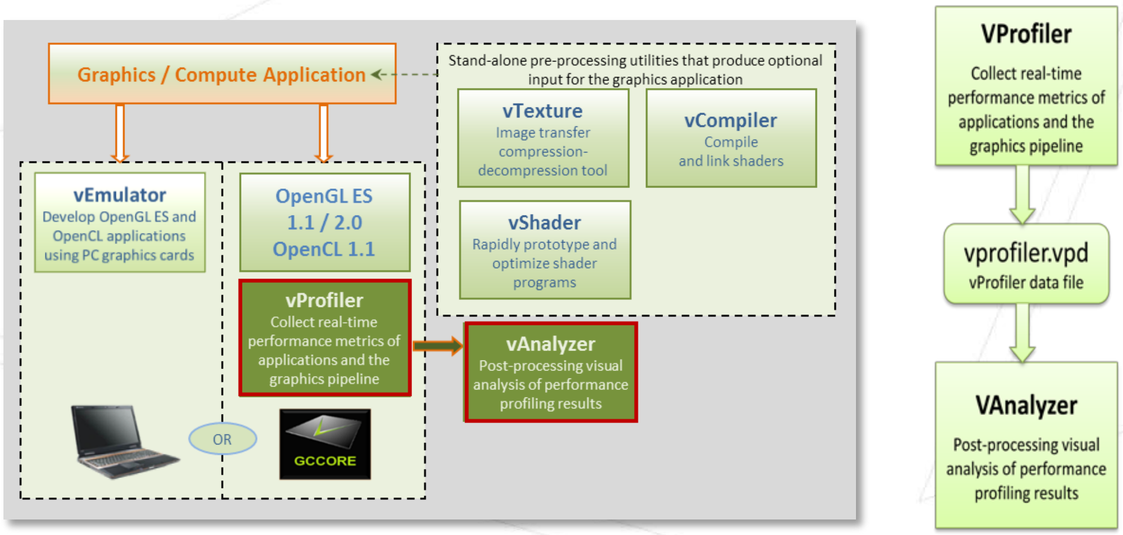

Graphics / Compute Application
Stand-alone pre-processing utilities that produce optional input for the graphics application
vTexture
Image transfer compression-decompression tool
vCompiler
Compile and link shaders
vShader
Rapidly prototype and optimize shader programs
vEmulator
Develop OpenGL ES and OpenCL applications using PC graphics cards
OpenGL ES 1.1 / 2.0 OpenCL 1.1
vProfiler
Collect real-time performance metrics of applications and the graphics pipeline
vAnalyzer
Post-processing visual analysis of performance profiling results
OR
GCCORE
VProfiler
Collect real-time performance metrics of applications and the graphics pipeline
vprofiler.vpd
vProfiler data file
VAnalyzer
Post-processing visual analysis of performance profiling results

Figure 8. vProfiler performance profiling save data for review in the vAnalyzer visual analyzer

6.1 Fundamentals of Performance Optimization

Whenever an application runs on a computer, it makes use of one or more of the available resources. These compute resources include the CPU, the graphics processor, caches and memory, hard disks, and possibly even the network. Viewed simplistically, it will always be the case that one of these resources will be the limiting factor in how quickly the application can finish its tasks. This limiting resource is the performance bottleneck. Remove this bottleneck, and application performance should improve. Note, however, that removing one limiting factor will always promote something else to become the new performance bottleneck.

The goal of optimizing, or tuning, application performance is to balance the use of resources so that none of them holds back the application more than any of the others. In practice there is no single, simple way to tune an application. The whole system needs to be considered, including the size and speed of individual components as well as interactions and dependencies among components.

vProfiler collects information on GPU usage and on calls to Vivante functions within the graphics pipeline. As such it provides an excellent view into what is happening on the GCCORE graphics processor at any point in time, down to the individual frame. So when your application performance is GPU-bound, vProfiler and vAnalyzer are the right tools to help you determine why.

Please note that the initial determination regarding which component of your computer system is the performance bottleneck—CPU, GPU, memory, etc.—is the domain of system performance analyzers and is outside the scope of the GPU tools. A list of such performance analysis tools can be found at Wikipedia: http://en.wikipedia.org/wiki/List_of_performance_analysis_tools.

6.2 vProfiler Setup for Linux

The VTK Windows package includes vAnalyzer for the Windows environment. The vProfiler tool can be compiled for Linux, as per the instructions below.

vProfiler stores software and hardware counters captured per frame in the **vprofiler.xpd** file. vAnalyzer reads the .vpd file and allows the user to browse all counters, visualize application performance bottlenecks, and measure system utilization of that application run. Presently, vProfiler does not store frame buffer images due to excessive overhead that changes the behavior of applications.

6.2.1 Building Drivers with vProfiler Option

When building Vivante Graphics Drivers in a Linux environment, please add the following build command option to enable vProfiler:

```
make -f makefile.linux USE_PROFILER=1
```

In some cases when GPU power management is enabled, vProfiler may not be able to get the correct GPU counters. So to ensure that the vProfiler works properly, GPU power management should be disabled when building the driver. The corresponding build options are specified as below.

```
make -f makefile.linux USE_PROFILER=1 USE_POWER_MANAGEMENT=0
```

6.2.2 Set vProfiler Environment Variables

The following table summarizes the environment variables that vProfiler supports.

Table 4. vProfiler Environment Variables

Environment Variable	Description
VPROFILER_OUTPUT	Specify the output file name of vProfiler
VPROFILER_FRAME_NUM	Specify the number of frames dumped by vProfiler
VPROFILER_SYNC_MODE	Enable or disable the synchronous mode of vProfiler

6.2.2.1 VPROFILER_OUTPUT

The output file of vProfiler is **vprofiler.vpd** by default. To specify an alternate filename use the environment variable **VPROFILER_OUTPUT**. For example,

```
export VPROFILER_OUTPUT = sample.vpd
```

6.2.2.2 VPROFILER_FRAME_NUM

The profile file generated when running an application for a long time can be very large. This takes up a large amount of disk space and also makes it hard to view the data in vAnalyzer. To limit the number of frames to analyze, use the environment variable **VPROFILE_FRAME_NUM**. For example, this example setting will make vProfiler dump performance data for the first 100 frames.

```
export VPROFILER_FRAME_NUM =100
```

6.2.2.3 VPROFILER_SYNC_MODE

To get accurate values from the GPU counters, vProfiler needs to commit the GPU commands at the end of every frame; this is so-called synchronous mode. The environment variable **VPROFILE_SYNC_MODE** can be used to enable or disable synchronous mode. By default, vProfiler works in asynchronous mode. The command below will make vProfiler work in synchronous mode.

```
export  VPROFILER_SYNC_MODE=1
```

6.3 vProfiler Collects Performance Data

vProfiler is implemented by utilizing hardware counters and a group of instrumentations inserted into drivers that are controlled by compilation flags.

6.3.1 Performance Counters

vProfile counters are divided into five sets: hardware, HAL (Vivante Graphics driver), (shader) program, OpenGL and OpenVG. The counters provide detailed per frame runtime information about the application that can help the developer monitor and tune an application's resource usage. The following table briefly lists the various profile counters. For further information, see Appendix A at the end of this document.

Table 5. Performance Counter Types

Counter Type	Description
HWCounters	A detailed profile of every stage of the GPU hardware pipeline
HALCounters	Driver memory usage
Program	Statistics of the shaders loaded in the GPU (Note: Available only for OpenGL ES 2.0 applications.)
OGLCounters	Various OpenGL (OpenGLES 20 or 11) counters, such as API usage and primitives drawn.
OVGCounters	Various OpenVG counters, such as API usage and primitives drawn.

6.4 vAnalyzer Viewing and Analyzing a Run-time Profile

vAnalyzer is a GUI-based tool whose purpose is to help the user view and analyze GPU performance data that was collected using counters during an application run. The performance data from a binary file(*.vpd) written by vProfiler is displayed by vAnalyzer both in text lists and as line graphs. vAnalyzer features a multi-tab, multi-pane, graphical user interface that gives the user several ways to inspect any frame in a captured animation sequence.

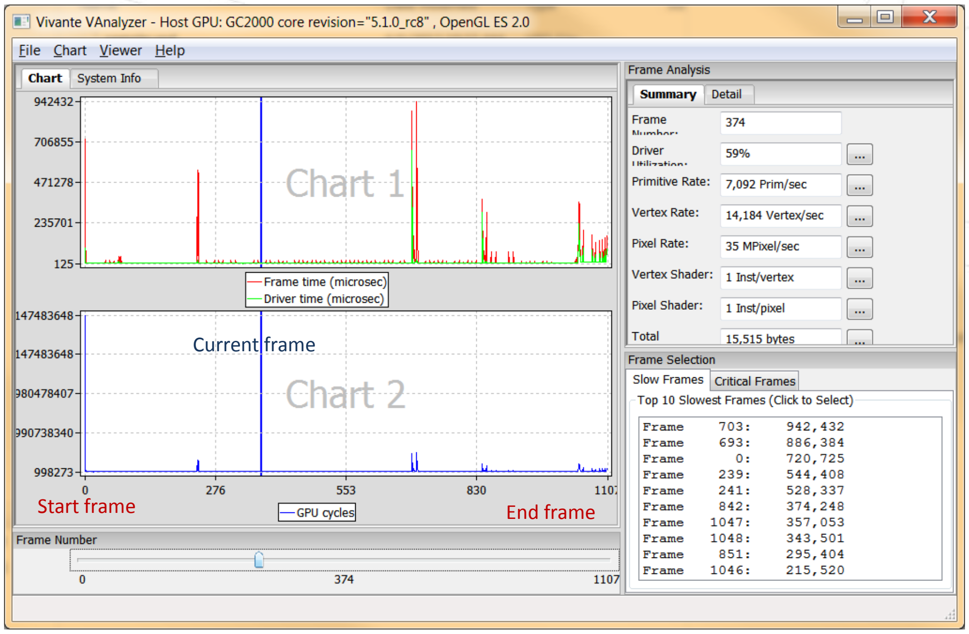
6.4.1 Loading Profile Files

vAnalyzer accepts a profile for input, which is a **.vpd** file of performance data created by the Vivante vProfiler during a run. For example, the saved file may have a name such as **sample.vpd**.

A **.vpd** file can be selected using the **File/Load Profile Data** menu option.

When a performance profile is loaded, vAnalyzer populates the title bar with information about the GPU and the CPU.

The vAnalyzer screen shot below shows the vAnalyzer GUI immediately after loading a .vpd performance file, and moving the frame number slider to frame 374. By default, the main pane of the vAnalyzer window will display the Charts tab which provides charts for frame time, driver time and GPU cycles. Additional charts can be added in the graph window by selecting from the list of variables on the right. Different combinations of counters can be displayed in graphical and list form to illustrate resource utilization for any portion of the profiled application. A second tab contains system information.



Frame Number:	374
Driver Utilization:	59%
Primitive Rate:	7,092 Prim/sec
Vertex Rate:	14,184 Vertex/sec
Pixel Rate:	35 MPixel/sec
Vertex Shader:	1 Inst/vertex
Pixel Shader:	1 Inst/pixel
Total	15,515 bytes

Top 10 Slowest Frames (Click to Select)

```
Frame    703:    942,432
Frame    693:    886,384
Frame      0:    720,725
Frame    239:    544,408
Frame    241:    528,337
Frame    842:    374,248
Frame   1047:    357,053
Frame   1048:    343,501
Frame    851:    295,404
Frame   1046:    215,520
```

Figure 9. vAnalyzer GUI Main Window

6.4.2 vAnalyzer Menu Bar

The vAnalyzer main window opens when a user launches vAnalyzer. The main menu bar contains drop-down menus for **File**, **Chart**, **Viewer** and **Help**. Menu options include the following:

File

- **Load Profile Data**: load a .vpd profile file
- **Export Current Frame Data**: dump all the counters for the frame being viewed to a .cvs file
- **Exit**: exit vAnalyzer

Chart

- **Create chart**: create a new chart
- **Customize chart**: add or delete counters in an existing chart
- **Remove chart**: delete a chart
- **Export data from chart**: dump the counters in a chart to a .csv file
- **Save chart to png** : dump the chart to a .png file
- **View**: zoom in, zoom out or fit the chart

Viewer

- **OpenGL function call viewer**: display the OpenGL function call statistics
- **Program viewer**: display the shader program statistics

Help

- **About**: gives version information for vAnalyzer

6.5 vAnalyzer Charts

6.5.1 vAnalyzer Upper Left Pane: Chart Tab and Menu Options

On the Chart tab in the vAnalyzer main window two default line graphs are displayed.

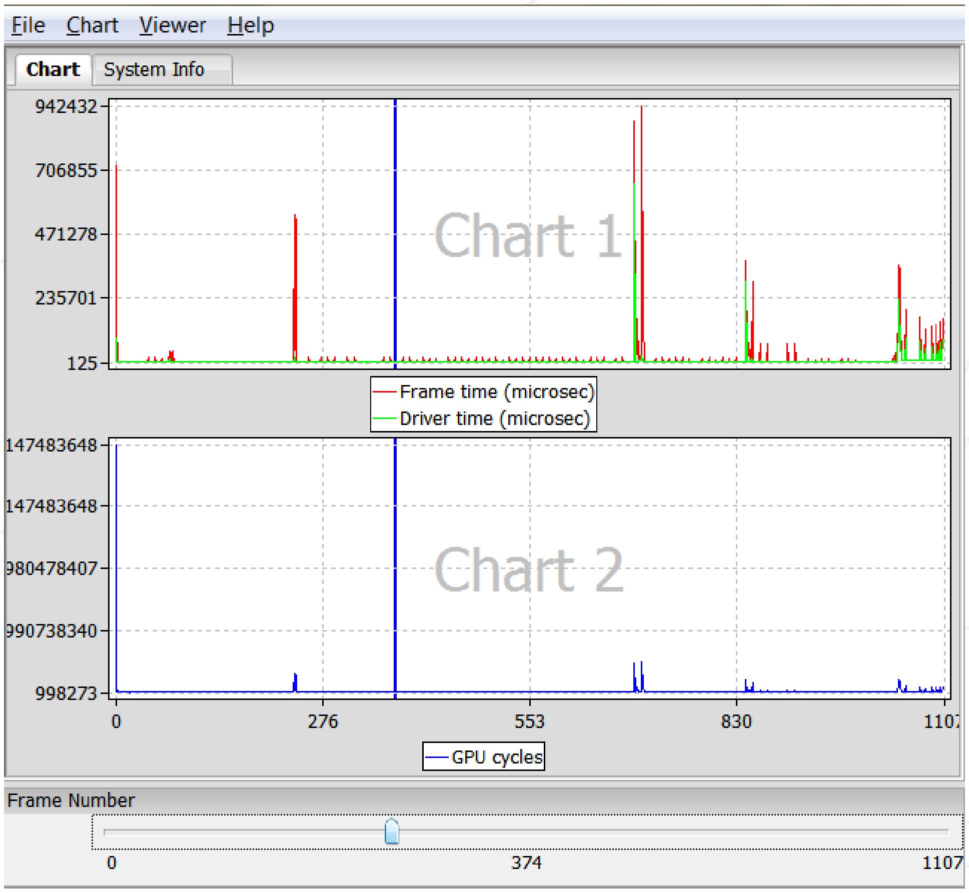


Figure 10. vAnalyzer Performance Counter Charts

6.5.2 Chart Customization

Chart / Customize Chart: Additional performance counters can be added to existing chart using the **Create New Chart** dialog window, which can be invoked from the drop menu **Chart / Customize Chart**, or from a pop-up menu which can be invoked by right clicking in the Chart tab area.

New Chart: A new chart can be added in a similar way. A single chart can display up to four (4) counters, and the Chart pane can hold up to eight (8) charts. Thus a maximum of thirty-two (32) counters can be graphed at the same time.

Remove Chart: Any chart can be removed from the display using the drop menu **Chart / Remove Chart**.

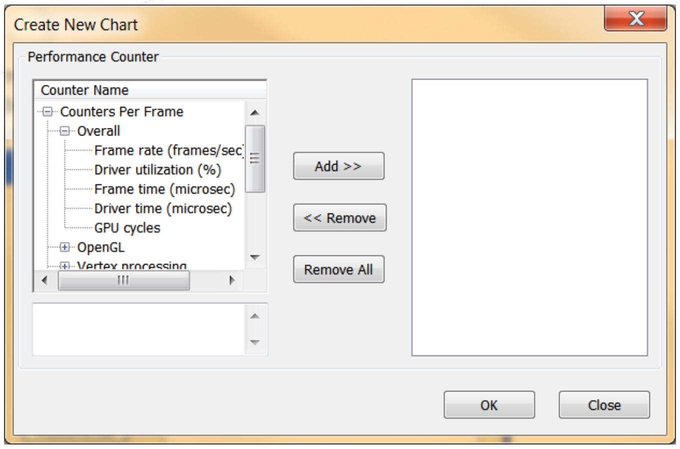


Figure 11. vAnalyzer Create New Chart Dialog

6.5.2.1 Chart Components and Navigation

Frame Marker: On the plots displayed in the chart example above there is a blue, vertical frame marker. This marks the current frame position in the timeline.

Zoom:

Zooming in on a set of frames can be achieved in one of two ways.

- One method is to hold down the left mouse button and then sweep a selection box across a range of frame numbers, either on a plot itself or in the common X-axis (frame numbers) in the "Chart" pane, before releasing the mouse button. All charts in the "Chart" pane will zoom in to the same range of frames.
- Alternatively, if your mouse has a scroll wheel, you may also zoom in by rolling the wheel forward--toward the screen.

To zoom out move the scroll wheel backward, toward you.

To reset zoom to the default, which will show the entire timeline, press the escape key (ESC) on the keyboard. The chart view will change to include all frames, from start to end.

6.5.2.2 Data Export

The performance counters in a chart can be dumped to a **.csv** file by selecting from the dropdown menu **Chart / Export data from chart**. The **.csv** file can be viewed using Excel or another text viewer.

The chart can also be dumped to a **.png** file by selecting from the main menu **Chart / Save chart to PNG**.

6.5.3 vAnalyzer Lower Left Pane: Frame Number Slider Bar

In the lower left pane of the vAnalyzer window, there is a Frame Number gauge in the form of a slider bar. Numbers at each end of the bar indicate the initial frame (0) and the last frame available in the loaded sample. By left-clicking and holding the slider, the user can change which frame is selected for analysis. When the frame number is changed, the blue vertical line which indicates the current frame will move, and the reported Frame Number will change in the upper right pane Frame Analysis Summary.

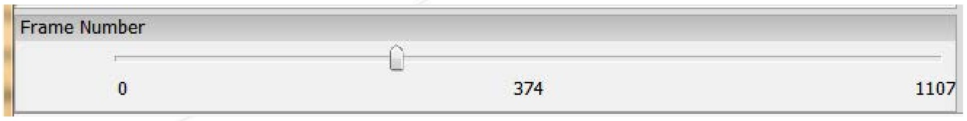
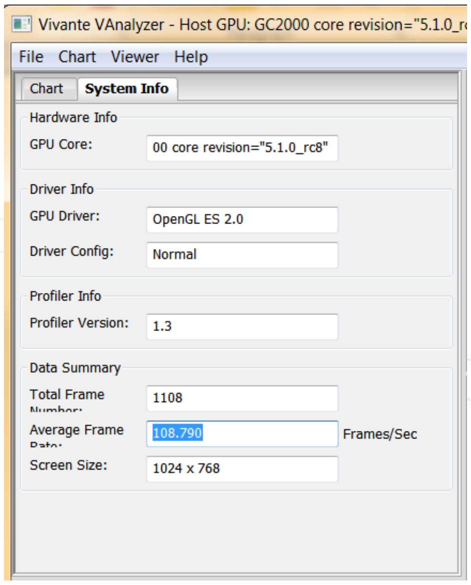


Figure 12. vAnalyzer Frame Number Slider Bar

6.5.4 vAnalyzer Left Pane: System Info Tab

When a **.vpd** profile is loaded, system information about the profiled machine populates the fields on the System Info pane. Some information is repeated in the title bar of the main GUI for quick reference.


Vivante VAnalyzer - Host GPU: GC2000 core revision="5.1.0_r

Hardware Info

GPU Core: 00 core revision="5.1.0_rc8"

Driver Info

GPU Driver: OpenGL ES 2.0

Driver Config: Normal

Profiler Info

Profiler Version: 1.3

Data Summary

Total Frame Number: 1108

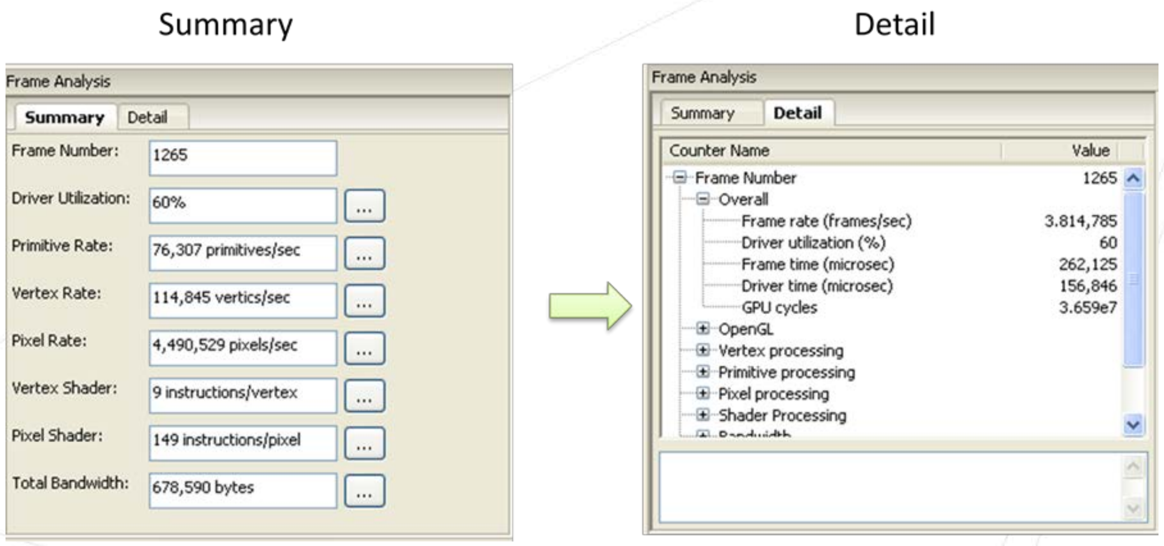
Average Frame Rate: 108.790 Frames/Sec

Screen Size: 1024 x 768

Figure 13. vAnalyzer System Info Tab

6.5.5 vAnalyzer Upper Right Pane: Frame Analysis

A selection of performance counters for the frame being viewed are displayed on the right side of the vAnalyzer main GUI. The user can convert this pane to a pop-up window by dragging the pane outside the application window. Drag it back to the right pane area of the application window to reintegrate the pane.



Frame Number:	1265
Driver Utilization:	60%
Primitive Rate:	76,307 primitives/sec
Vertex Rate:	114,845 vertics/sec
Pixel Rate:	4,490,529 pixels/sec
Vertex Shader:	9 instructions/vertex
Pixel Shader:	149 instructions/pixel
Total Bandwidth:	678,590 bytes

Counter Name	Value
Frame Number	1265
Overall	
Frame rate (frames/sec)	3.814,785
Driver utilization (%)	60
Frame time (microsec)	262,125
Driver time (microsec)	156,846
GPU cycles	3.659e7
OpenGL	
Vertex processing	
Primitive processing	
Pixel processing	
Shader Processing	

Figure 14. vAnalyzer Frame Analysis Summary and Detail Tabs

6.5.5.1 Summary Tab

The **Summary** tab displays summary information for the frame being viewed.

The Selected Frame Number can be changed by entering a new frame number in the text box at the top of the list. The user must press Enter after the input to activate the change. Then Summary values, sliders, and charts all change to reflect the newly entered frame number.

The Summary values below frame number are not directly changeable. They change only when the frame number is changed, either in the Summary tab, by moving the Frame Number slider, or by selecting a frame from the Frame Selection pane. Clicking the "**…**" button to the right of a **Summary** item will bring up the corresponding counters in the **Detail** tab. For example, clicking the "**…**" button to the right of **Primitive Rate:** switches the view to the **Detail** tab and expands the **Primitive processing** catogory. Clicking the "**…**" button for **Driver Utilization:** brings up the pop-p window **OpenGL function call viewer**.

6.5.5.2 Detail Tab

The **Detail** tab reports values for overall performance evaluation, such as Frame Rate, Driver Utilization, and GPU cycles. Additionally counter detail is accessible on this tab. The categories of available counters in the **Detail** tab are: Overall, OpenGL, Vertex processing, Primitive processing, Pixel processing, Shader Processing and Bandwidth. Appendix A lists performance as well as hardware counters.

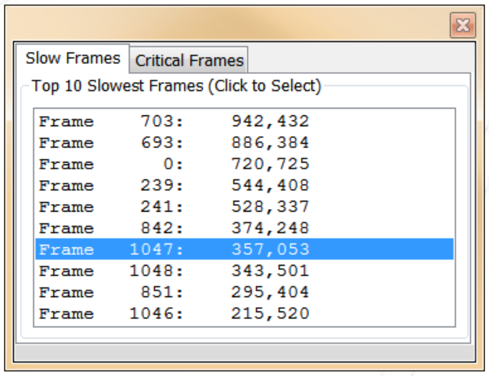
6.5.6 vAnalyzer Lower Right Pane: Frame Selection

As with the Frame Analysis pane, this pane can be dragged to display as an independent popup window.

6.5.6.1 Slow Frames Tab

The "Slow Frames" tab lists the ten (10) slowest frames in the animation sequence, by time in ascending order from slowest to tenth slowest.

The user can left-click on any entry, or can use the arrow keys to move up and down the list, and the display in each of the other GUI panes will change to match that frame.



Top 10 Slowest Frames (Click to Select)

```
Frame     703:     942,432
Frame     693:     886,384
Frame       0:     720,725
Frame     239:     544,408
Frame     241:     528,337
Frame     842:     374,248
Frame    1047:     357,053
Frame    1048:     343,501
Frame     851:     295,404
Frame    1046:     215,520
```

Figure 15. vAnalyzer Frame Selection Slow Frames Tab

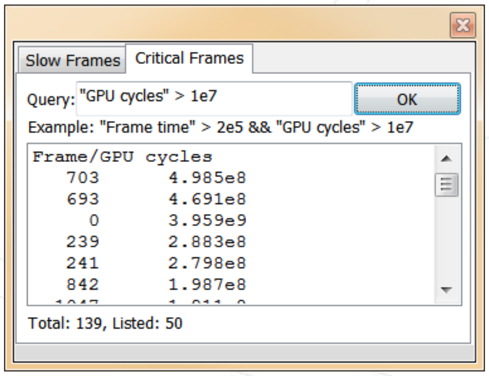
6.5.6.2 Critical Frames Tab

Select the "Critical Frames" tab to customize the criteria by which a frame is chosen for inspection. One or more of the performance counters can be specified in building the query, which also allows for AND and OR logic.

Queries should follow a pattern such as:

`"counter name" condition('<'/'>'/'==') values.`

Users can identify counter names from those in the Frame Analysis pane Detail tab. An example is provided just below the Query input text box.



Query: "GPU cycles" > 1e7

Example: "Frame time" > 2e5 && "GPU cycles" > 1e7

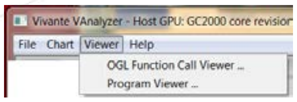
```
Frame/GPU cycles
   703       4.985e8
   693       4.691e8
     0       3.959e9
   239       2.883e8
   241       2.798e8
   842       1.987e8
```

Total: 139, Listed: 50

Figure 16. vAnalyzer Frame Selection Critical Frames Tab

6.6 vAnalyzer Viewers

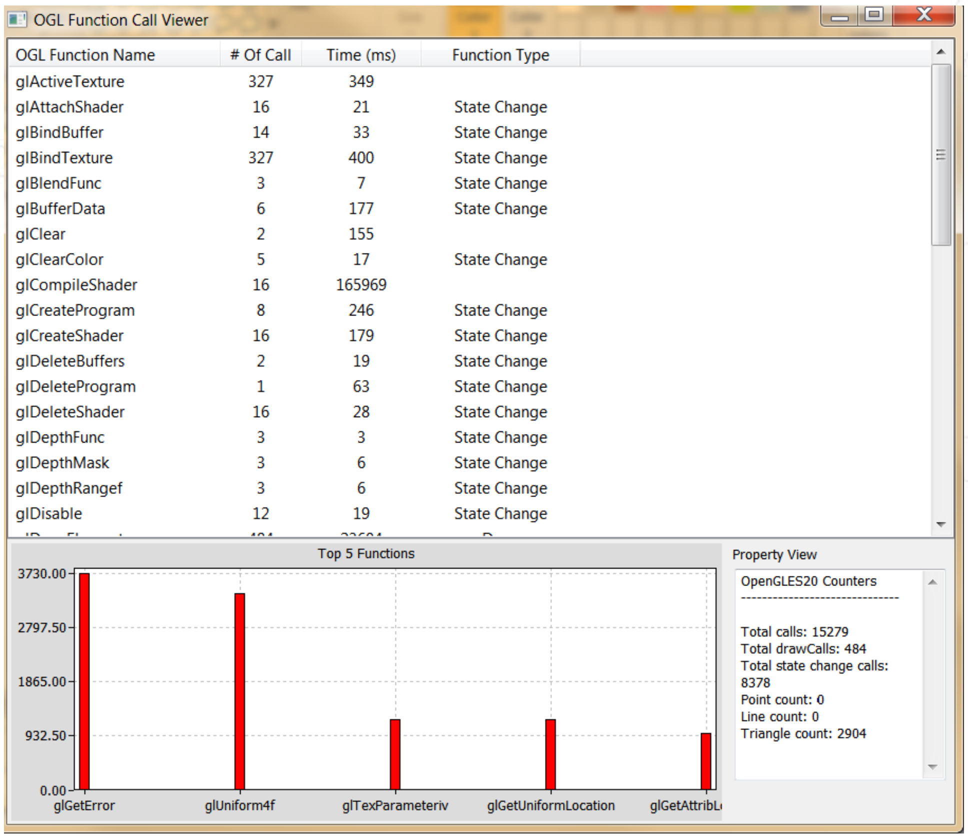
The Viewer information pop-up windows can be launched by selecting **Viewer/OpenGL function call viewer** or **Viewer/Program viewer** from the Main menu. The selected Viewer will appear in a pop-up window.


Vivante VAnalyzer - Host GPU: GC2000 core revision
File
Chart
Viewer
Help
OGL Function Call Viewer ...
Program Viewer ...

6.6.1 OpenGL Function Call Viewer

The OpenGL function call viewer includes three information areas.

- The **OGL Function Name** area contains a table which lists the available OpenGLES20 or OpenGLES11 functions by Function Name and Function Type, the run time and the number of times each has been called for this frame. Functions can be sorted by clicking in the column heading area. For example, you can sort the functions by call count or run time by clicking the title bar of "# of Call" or "Time (ms)".
- The **Top 5 Functions** area contains a histogram which shows the top 5 call count of the listed OpenGL functions.
- The **Property view** area shows the summary when no function is selected; while it shows performance hints for the function when one is selected.



OGL Function Name	# Of Call	Time (ms)	Function Type
glActiveTexture	327	349	
glAttachShader	16	21	State Change
glBindBuffer	14	33	State Change
glBindTexture	327	400	State Change
glBlendFunc	3	7	State Change
glBufferData	6	177	State Change
glClear	2	155	
glClearColor	5	17	State Change
glCompileShader	16	165969	
glCreateProgram	8	246	State Change
glCreateShader	16	179	State Change
glDeleteBuffers	2	19	State Change
glDeleteProgram	1	63	State Change
glDeleteShader	16	28	State Change
glDepthFunc	3	3	State Change
glDepthMask	3	6	State Change
glDepthRangef	3	6	State Change
glDisable	12	19	State Change

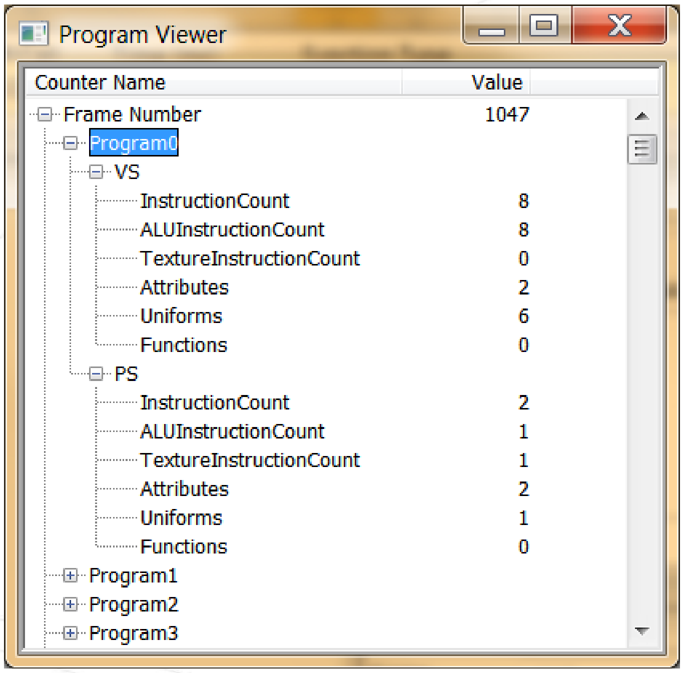
OpenGLES20 Counters

Total calls: 15279
Total drawCalls: 484
Total state change calls: 8378
Point count: 0
Line count: 0
Triangle count: 2904

Figure 17. vAnalyzer OpenGL function call viewer window

6.6.2 Program Viewer

For a given Frame Number, the Program Viewer gives the statistics for shader programs: uniforms, attributes, and the number of instructions in the shader. This is only for OpenGLES20 profile data.



Counter Name	Value
Frame Number	1047
Program0	
VS	
InstructionCount	8
ALUInstructionCount	8
TextureInstructionCount	0
Attributes	2
Uniforms	6
Functions	0
PS	
InstructionCount	2
ALUInstructionCount	1
TextureInstructionCount	1
Attributes	2
Uniforms	1
Functions	0
Program1	
Program2	
Program3	

Figure 18. vAnalyzer Program Viewer

Appendix A: Debug and Performance Counters

Hardware Counters (listed by sub-block)

Top Level

- Total cycles
- Outstanding Reads
- Outstanding Writes
- Read bandwidth
- Write bandwidth
- Total bandwidth

Host Interface

- Number of cycles AXI read request is stalled
- Number of cycles AXI write request is stalled
- Number of cycles AXI write data is stalled

Memory Controller

- Total 8 byte read requests from pipeline
- Total 8 bytes read requests from the core
- Total 8 byte write requests from pipeline

Primitive Assembly

- Total vertex count
- Input primitive count
- Output primitive count
- Depth clipped primitive count
- Trivial rejected primitive count
- Face culled primitive count

Front End Vertex Processing

- Input vertex count
- Vertices per batch
- Vertices per primitive

Setup

- Culled triangles
- Culled lines

Pixel Engine

- Pixels killed by color pipe, % alpha test fail
- Pixels killed by depth pipe, % depth & stencil test fail
- Pixels drawn by color pipe
- Pixels drawn by depth pipe
- Valid pixel count
- Overdraw

Shader

- Total shader cycle count
- Vertex shader active cycles
- Vertex shader idle cycles
- Pixel shader active cycles
- Pixel shader idle cycles
- Total vertex instructions executed
- Total vertices shaded
- Total pixel instructions executed
- Total pixels shaded

Raster Unit

- Valid pixels
- Total quads (after EEZ)

- Valid quads (after early-z and EEZ)
- Total primitives
- Cache misses (in the pipeline)
- Cache misses (in the pre-fetcher)
- EEZ culled quads

Texture Unit

- Total bilinear texture requests
- Total trilinear texture requests
- Total texture requests
- Total discarded texture requests
- Memory read count
- Memory read count in 8 byte
- Cache misses (in the pipeline)
- Total hitting texels (in pre-fetcher)
- Total missing texels (in pre-fetcher)

HAL Counters

Index Buffer, Texture Buffer, and Vertex Buffer

- New bytes allocated
- Total bytes allocated
- New object allocated
- Total objects allocated

Overall Computed Values

- Frame rate
- Driver utilization
- Frame time
- Driver time
- GPU cycle

Shader Processing Counters

Vertex Shader and Fragment Shader

- Total instruction count
- ALU instruction count
- Texture instruction count
- VS branch instruction count
- VS texture fetches
- PS branch instruction count
- PS texture fetches
- Function calls
- Attribute count
- Uniform count

3D API Counters (OpenGL ES, D3D, etc.)

Vivante generated

- Number of lines drawn
- Number of points drawn
- Number of triangles drawn
- Shader compiler time
- Total OpenGL function calls
- Total OpenGL draw calls
- Total OpenGL state change calls

OpenGL API Call List

- gl* Note: This list can be generated by combining the function names listed in the following two locations.
  OpenGL ES 2.0 Reference Pages at Khronos.org: http://www.khronos.org/opengles/sdk/docs/man/
  OpenGL ES 1.1 Reference Pages at Khronos.org: http://www.khronos.org/opengles/sdk/1.1/docs/man/

Document Revision History

Version	Date	Compatible product	Notes
1.2	30 October 2012	SW release 4.6.9.p9 VTK v1.4.2	Remove VDK and vdkEmulator references. Update Table 1 Directory Contents Revise Toolkit Diagram (Figures 1,2,3,4, 8,9,10)
1.1	11 October 2012	SW release 4.6.9 VTK v1.4	Remove Confidential Watermarks. Update Copyright and contact notices. Revise Toolkit Diagram (Figures 1,2,3,4, 8,9,10)
1.0	14 June 2012	SW release 4.6.9	Renamed from Vivante SDK User Guide (v1.4)to Vivante Tool Kit User Guide 1.0 Updated vProfiler and vAnalyzer, add vTexture, miscellaneous refinements.