# Video Test Suite SDK

## Users and Developers Guide

**RHMG Software Tools Library**

**5/25/2013**


**Authors:** Bill and Scott Werba
**Developer:** Scott Werba


**Document ID:** 3000-000
**Revision:** v1.1.0
**Publication Date:** May 25, 2013
**Publication Status:** Final

# 1.0 Introduction

As part of our own HD film transfer system development, we needed a means to verify and measure the digitized video signal and colour levels directly from our HD imager. Developing a suite of video measurement tools that was integrated into our HD software interface helped define the level of performance of our imaging system.

We developed a software based Waveform Monitor, a Vector scope, a Histogram and a YUV/RGB parade video instrument set. Now in DLL form, the video measurement toolset can be integrated into your own video project, using standard software development tools and practices. We used Visual Studio 2005 for base development and updated it in Visual Studio 2010.

The colour space conversion for each instrument follows industry SD and HD video standards BT.610 for SD and BT.709 for HD (depending on mode set). Originating colour space is pure 24bit RGB with full gamut and converted to limited gamut YUV or YCrCb depending on the user setting for each instrument, with appropriate 8 bit scaling and IRE level limits.

Source data for each instrument can be from standard uncompressed RGB based bitmap files or from RGB values originating from a pixel data array in memory. RGB order must first be defined.

The software library comes with a number of DLL functions that allows rendered output to be displayed in OpenTK GLControl. This version of our release is .Net Framework 3.5 based. Multiple instances can also be incorporated if required, each with its own data source and process and GL Control for render, assuming each instance has been setup independently with its own variables.

The software requires Framework 2.0 and Framework 3.5 for these DLLs to operate. The only hardware consideration is that there is a graphics card in your computer system that supports OpenGL v1.0 at a minimum. That's it.

Each software video instrument offers various setup and modes of operation that are described in this document. The C# sample code shows how to setup and code the DLL's into a typical application.

Due to the fact when rendering an HD image the data size of 1920x1080 is quite large, there is the ability to decimate the rendered image to allow a recovery in refresh speed. This can be done automatically (LOD function) or manually. The quality of the output of a decimated display will only be evident if the factor is set too high. Decimation factor is an integer value which affects the number of lines output vertically. Lines are skipped uniformly in factors of 2 and output for display. The more the decimation factor the less accurate the display and perhaps the more rough looking output.

*DLL System Configuration*
The DLL internal operation is to decode the input arguments, to act on them and to calculate the necessary data output to be displayed. They in of themselves have no display capabilities. That is done outside the DLL using GLControl on a form. After the relevant input data has been processed, the resulting data is rendered internally to meet the selected instruments display output properties. The DLL will then supply an output map of the rendered data to a GLControl.

# 1.0 Installation of SDK

**To Install**

1) Run the file "SDK_Setup.exe" to a directory of your choice. Upon extraction this directory will contain a number of DLLs, a readme file, an OpenTK license, and a C# source code example.

2) In Visual Studio import the DLLs in the Solution explorer and add in the References; OpenTK, OpenTK.GLControl, QuickFont, and ScopeTools

3) Add; using OpenTK, using OpenTK.GLControl, using ScopeTools and using BitmapHeader to the form that will be using the Scope Tools.

4) Declare a ModularScopeDisplay or a Scope_??? Variable type

5) On initialization have the variable assigned to a new of its type with a ref of the GLControl on the form, and ScopeMode if it has one.

6) Setup the Variable's Init, SetupProjection and SelectConfig Functions with the proper data. And when ready to render, use the Render function to render

**Important note:**
You must include the all DLLs, the readme file and the OpenTK license when you distribute the product.

# 3.0 Video Toolset Description

## 3.1 Waveform Monitor

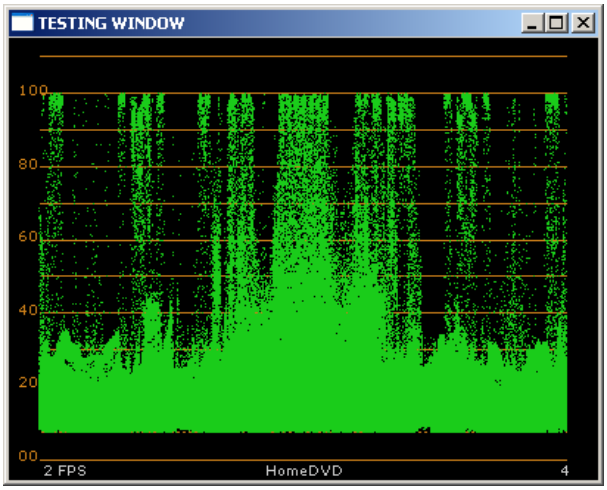


**Figure 3.0 Waveform Monitor (in IRE) Reference Image**

The *Waveform Monitor* is an industry standard video calibration instrument used to measure the video luminance and chroma components in a colour image or a video signal. Video can be represented in digital data or an analog signal form.

Newer WFM technology do not have analog inputs anymore but use digital SDI type serial input for SD and HD video display work. In *Figure 3.0* above, shows a typical luminance output of an RGB based image frame. The RGB values of the input file or array are converted to YCrCb and only the Y component is used for display purposes. Y calculations are based on either SD or HD selected colour spaces. Right now we do not support the chroma component for the WFM display.

The equations used within this application are:

SD Eqn* 65.738R + 129.057G +25.064B + 16
HD Eqn* (46.742/256)R + (157.243/256)G + (15.874/256)B + 16

*SD Ref: en.wikipedia.org/wiki/ycbcr
*HD Ref: Charles Poyton Errata - Digital Video and HDTV Algorithms and Interfaces

The smallest window or picture box this function can be displayed in is 342 by 240 pixels, before any annotated text in the rendered display will disappear.

### 3.1 VectorScope

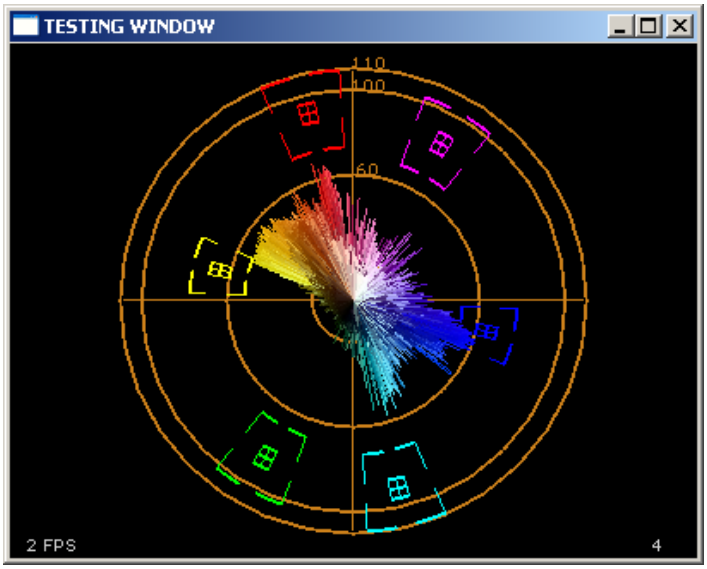


**Figure 3.1 Vector scope, RGB mode with Line and Color options**

The Vector scope is a video tool used to see the Hue and Saturation levels of a colour image source. It is a great tool for indicating white balance of an input image, as shifts in hue are readily shown when offset from the white and black center of the scope. The chrominance component of an image can be isolated when converting the RGB colour space to YCrCb and only using the CrCb component and calculating the necessary vector (angle and magnitude) from the colour data. RGB data can be sourced from a file or from an array of data which is user selectable. Image Vector Scope indicated above in *Figure 3.1* shows the vector results of the same reference image shown in *Figure 3.0*.

The Vector scope can display the rendered data in terms of Percent, IRE or RGB. The user can also select the rendered result in direct colour association or in standard instrument green colour. In addition the traces can be displayed as dots or lines.

The smallest window or picture box that can be programmed for this instrument is 280 by 280 pixels. Have the aspect ratio be as close to 1:1 as possible. Using a 4:3 AR is the next best standard to use if necessary.

The targets for the primary colors and their complements are to specification. The mini-targets are at 2.5 IRE and at 2.5 degrees. The outer targets are at 20 IRE and 10 degrees. The layout rings are there to help the user with finding the value of the visual representation plotted relative to the center of the display which is white/black levels (255 and 0).

### 3.2 RGB Histogram

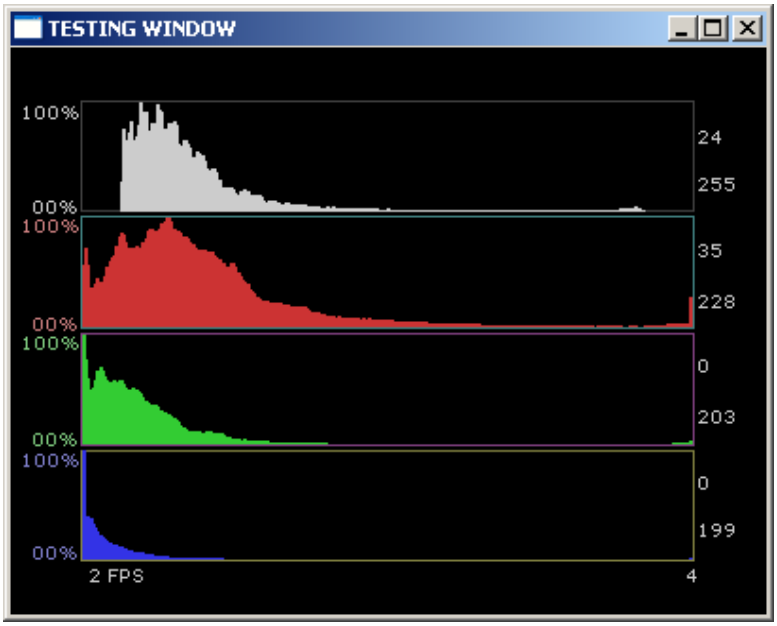


**Figure 3.2 Histogram**

The Histogram is a simple statistical tool used to view the number of occurrences of a particular pixel value in an image. Pixel depth is from 0 to 255 and noted in the x-axis and the number of occurrences for any one pixel is normalized in the Y axis.

The Histogram accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data sourced pointers. The system can display the data only in Percent right now, and finds the most common and least common color value in the image. The smallest window or picture box this tool can use is 350 by 160 pixels.

### 3.3 RGB Parade

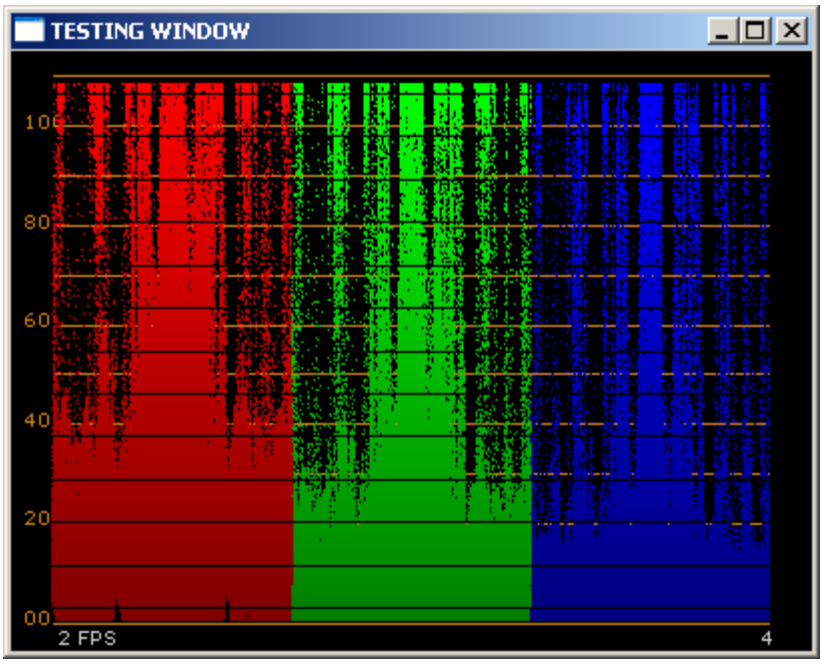


**Figure 3.3 RGB parade, RGB mode and Tri-mode**

The RGB parade is a tool used to see the individual RGB color values and intensities in an image.

The RGB parade accepts 24-bit RGB bitmap files, and RGB, RGBA, BGR and BGRA data pointers. The system can display the data in Percent, IRE and RGB. The user can set the tool to only show RGB (Tri-mode) or show luminance and RGB (Quad-Mode). The smallest window or picture box this tool can use is 342 by 240 pixels.

### 3.4 YUV Parade

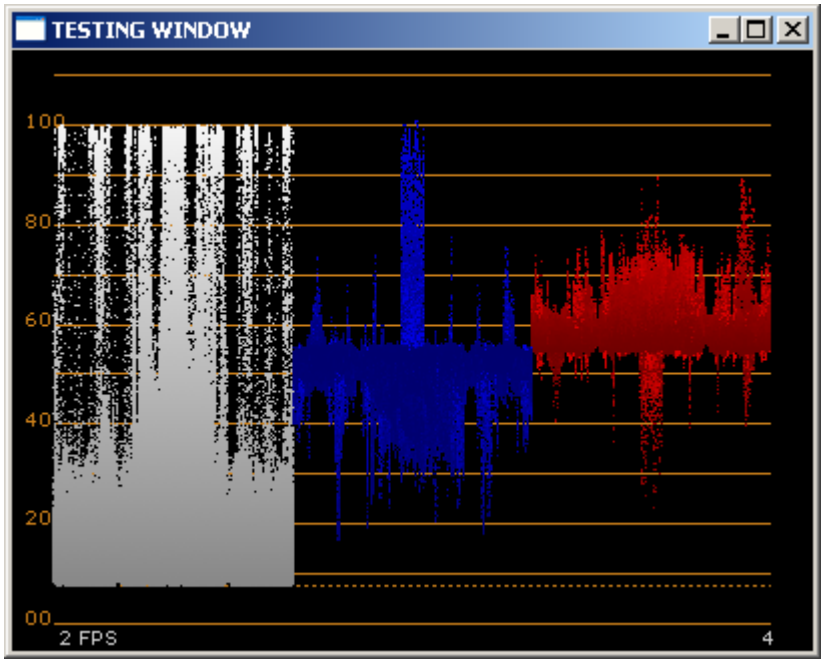


**Figure 3.4 YUV Parade**

The YUV parade is a tool used to see the luminance and chrominance values in an image.

The YUV parade accepts 24-bit RGB bitmaps, and RGB, RGBA, BGR and BGRA data pointers. The system can display the data in Percent, IRE and RGB. The smallest window or picture box this tool can use is 342 by 240 pixels.

# 4 DLL Description

## 4.1 Introduction

There are eleven function calls in this DLL set. Most are mandatory and must be run in the order shown in *Section 4.4*. Some are optional. The optional calls are meant to be part of tight loop to provide best performance. Using the LOD component(s) in your code for example, will optimize the best decimation value for the images input and thus the highest rendering speed for the computer platform you are using. High performance computer platforms will offer high speed rendering and allow other multitasking operations without or minimal impact on the quality of the render.

## 4.2 Functions Description

Every instance of a scope requires an instance of a GLControl that it is to renders on.

The Contructor of any scope requires a reference to a GLControl that it'll render to, The ModularScopeDisplay has a second variable which configures it to a specific scope type using an enum ScopeMode

An example of using ScopeBase:

```
sBase = new Scope_WFM(ref GL_Display);
```

An example of using ModularScopeDisplay:

```
MSD1 = new ModularScopeDisplay(ref GLDisplay1, ScopeBase.ScopeMode.WFM);
```

There are two required setup functions; `SetupProjection and Init.`

Init requires the size / resolution of the visual data that it will be processing to accurately render the results of the information SetupProjection requires the size of the GLControl it'll be rendering to, to make correct rendering aspect ratio and FOV. If there is a change to either one the scope needs to be updated with the new information or it'll mis-render the data or worse crash.

An example of using ScopeBase

```
sBase.Init(1920,1080);
sBase.SetupProjection(GLDisplay1.Size.Width, GLDisplay1.Size.Height);
```

An example of using ModularScopeDisplay

```
MSD1.Scope.Init(1920,1080);
MSD1.Scope.SetupProjection(GLDisplay1.Size.Width, GLDisplay1.Size.Height);
```

The scopes are configured using the SelectConfig function as this instructs the scope to be configured in a specific way when it's processing and rendering the result. The '*selection*' requires all 4 basic flags types* ORed together. Each instrument will require its own Config. However, the Vectorscope and RGB parade displays have extra flags that can be ORed in with the 4 basic flags types* shown below to give different ways of operating or rendering. See more detail on these flags in the Appendix A.

*The basic flags types are:

| | | |
|---|---|---|
| GL_DRM_xxx, | Data Read Mode | Where xxx can be BMP |
| GL_LAC_xxx, | Layout Control | Where xxx can be IRE |
| GL_LOD_xxx, | Level of Detail | Where xxx can be HI, MED or Low |
| GL_LUM_xxx | Colour Space | Where xxx can be YUV, HD, SD |

An example of using ScopeBase

```
sBase.SelectConfig(GL_DRM_BMP|GL_LAC_IRE|GL_LOD_MED|GL_LUM_YUV);
```

An example of using ModularScopeDisplay

```
MSD1.Scope.SelectConfig(GL_DRM_BMP|GL_LAC_IRE|GL_LOD_MED|GL_LUM_YUV);
```

You can tell the scope to include in the rendering how fast it's going by feeding it a string containing the numerial FPS of how fast it's rendering

An example of using ScopeBase

```
sBase.SetFPS(FPS.ToString());
```

An example of using ModularScopeDisplay

```
MSD1.Scope.SetFPS(FPS.ToString());
```

Since you may want optimal performance from the scopes you can call the AutoLOD function which will adjust the LOD (Level Of Detail) of the rendering so you get as much detail as possible with out the system dropping frames. The function though optional, is very useful if you wish to optimize the rendering speed of your project, since it calculates the best decimation factor to the image being rendered based on the performance level of your computer platform.. The range of decimation varies depending on operating mode. If set manually the range is from none to 2048, if in auto mode, none to 16K or better in factors of 2 (for very large images).

For example if the decimation factor is set to 2, and you have a 1920 x 1080 image, every other line is skipped. The horizontal pixel count stays intact for this version of the SDK.

Use the applicable function call (VB or C++) based on the coding language you are using.

An example of using ScopeBase

```
sBase.AutoLOD(FPS.ToString());
```

An example of using ModularScopeDisplay

```
MSD1.Scope.AutoLOD(FPS.ToString());
```

The Render Function will take the data and render the results based on the parameters set by the programmer or user. The function renders the bitmap RGB file or RGB data array values once and displays the rendered data according to the configuration and other parameters that were previously set. If there are new renders to run, use the function in a loop.

There are no timing constraints between the time a filename is changed to the time the render call is made. The function will merely skip the current render and come back again to run a render when the filename itself is stable. Render is the only function that has variation of data input, in that it'll take Bitmaps by file name, or you can use a variable of Memorystream or Byte Array without a header

An example of using ScopeBase

```
sBase.Render("BMPFileName.bmp");
sBase.Render(MemoryStream);
sBase.Render(Byte[]);
```

An example of using ModularScopeDisplay

```
MSD1.Scope.Render("BMPFileName.bmp");
MSD1.Scope.Render(MemoryStream);
MSD1.Scope.Render(Byte[]);
```

### 4.3 Flag Description

There are four basic types of flags; A Data Read Mode (DRM), a Layout Control (LAC), a Level Of Detail (LOD), and a Luminance (LUM) flag.

Refer to Flag *Appendix A* for more detailed info.

The **Data Read Mode** flag controls how to read the image data in and how to process the it. The "GL_DRM_BMP" flag tells the system that it is going to read 24-bit RGB bitmap, while flags that begin with "GL_DRM_" and end in"_PNT" informs the system that it will read a data pointer and the order of the values in the data pointer.

The **Layout Control** flag controls the layout and unit of measurement in the display. Some session types (VTR and RGB) have extra layout flags that change how the rendering system displays the data. The "GL_LAC_PER" uses a percent scale, "GL_LAC_RGB"uses the accual value of the data displayed, while "GL_LAC_IRE" displays the values in an IRE scale. An Example of the extra flags in the Vectorscope, if you just use the four basic flags and or includes the "GL_LAC_GRN" (Use green color) and "GL_LAC_PNT" (render Points) flags the display will use green dots to show the value of the data, but if you use "GL_LAC_CLR" (use the color value) and "GL_LAC_LIN" (render lines), while RGB Parade has two optional Flags GL_LAC_TRI (Tri-mode: RGB only) and GL_LAC QUAD (Quad-mode: Luma and RGB) with Tri-mode as a default if the flags are not used.

**Level Of Detail** is a must if a user does not want to use the AutoLOD function as this flag controls how many vertical lines it will skip, is displayed in the lower right corner. The value of 1 means it does every line, a 2 means every other line, 4 means every $4^{th}$ line and etc.. All these flags begin with "GL_LOD_", with "GL_LOD_PERF" telling the system to use every vertical line in the image and "GL_LOD_NPERF" doing every other line.

The **Luminance** flag controls the equations used in all calculations involving Luminance, Hue and Saturation that get displayed in screen. In other word this controls the color space equations used. The "GL_LUM_YUV" flag uses the YUVColor space, the "GL_LUM_STD" flag uses the Standard Definition color space, and the "GL_LUM_HID" flag uses the High Definition Color space.

### 4.4 Order of Functions

The order of the functions are listed below including optional functions.

1) Create a new variable of the scope

2) Initialize it
3) Setup the SetupProjection

4) Enter the SelectConfig

   ->Loop start

5) Use the SetFPS (optional)

6) Use the AutoLOD (optional)

7) And Render

   ->Loop end

The above order is the most optimal setup for configuring your program. You can change the order within certain limits to suit your needs, but be sure to use all the setup functions and clean up functions to operate the program well and without memory leaks.

Once the initialization functions have been executed they need not be run again. Only the Render functions need be called for fresh data to crunch and render.

### 4.5 Sample Code

Sample code is provided as part of this SDK package. The sample code for this version is supplied as C# code. Using Visual Studio 2010, the C# syntax and code constructs are near identical.

## 5 Minimum System Requirements

The program can work in any windows OS with Framework 2.0 and 3.5. The only hardware you need to have is a graphics card with OpenGL 1.0 or better.

## 6 Supported Development Environments

The program was made with Visual Studio 2005 and updated using VS 2010, OpenTK, and switching to C# language.

## 7 Support and Contact Information

No technical support will be available for this product at this time. The supplied documentation will describe as much detail as needed for a great percentage of users.

Support will only be provided for pre-sales questions

## 8 License

There is no software license attached to this toolkit or open source restrictions. You can do what you want with it. But you must follow the OpenTK license and restrictions.

I would however, appreciate that you don't share or give away this SDK to others without them paying for it through my web site. It's not that expensive.

# Appendix A

| Flag Name | Flag Type | Flag Value (Hex) | Function / Purpose |
|---|---|---|---|
| GL_DRM_BMP | Data Read Mode | 0x1000 | Configure system to read bitmaps |
| GL_DRM_RGB_PNT | Data Read Mode | 0x2000 | Configure system to data pointer of RGB format |
| GL_DRM_BGR_PNT | Data Read Mode | 0x3000 | Configure system to data pointer of BGR format |
| GL_DRM_RGBA_PNT | Data Read Mode | 0x4000 | Configure system to data pointer of RGBA format |
| GL_DRM_BGRA_PNT | Data Read Mode | 0x5000 | Configure system to data pointer of BGR format |
| GL_LAC_PER | Layout Code | 0x0100 | Set the Layout to Percent |
| GL_LAC_IRE | Layout Code | 0x0200 | Set the Layout to IRE scale |
| GL_LAC_RGB | Layout Code | 0x0300 | Set the Layout to char scale (0-->255) |
| GL_LAC_PNT | Layout Code VTR | 0x0000 | Configure the Vector scope to use Points |
| GL_LAC_LIN | Layout Code VTR | 0x0400 | Configure the Vector scope to use Lines |
| GL_LAC_GRN | Layout Code VTR | 0x0000 | Configure the Vector scope to use the Color Green |
| GL_LAC_CLR | Layout Code VTR | 0x0800 | Configure the Vector scope to use the color is is. |
| GL_LAC_TRI | Layout Code RGB | 0x0000 | Configure the RGB parade to display only RGB. |
| GL_LAC_QUAD | Layout Code RGB | 0x0800 | Configure the RGB parade to display Luma and RGB |
| GL_LOD_PERF | Level Of Detail | 0x0000 | Sets the LOD to perfect, no skipping |
| GL_LOD_NPERF | Level Of Detail | 0x0010 | Sets the LOD to near perfect, Does every other line |
| GL_LOD_VHIGH | Level Of Detail | 0x0020 | Sets the LOD to Very High, Does every 4th line |
| GL_LOD_HIGH | Level Of Detail | 0x0030 | Sets the LOD to High, Does every 16th line |
| GL_LOD_NHIGH | Level Of Detail | 0x0040 | Sets the LOD to Near High, Does every 32th line |
| GL_LOD_MEDH | Level Of Detail | 0x0050 | Sets the LOD to Medium High, Does every 64th line |
| GL_LOD_MED | Level Of Detail | 0x0060 | Sets the LOD to Medium, Does every 128th line |
| GL_LOD_MEDL | Level Of Detail | 0x0070 | Sets the LOD to Medium Low, Does every 256th line |
| GL_LOD_NLOW | Level Of Detail | 0x0080 | Sets the LOD to Near Low, Does every 512th line |
| GL_LOD_LOW | Level Of Detail | 0x0090 | Sets the LOD to Low, Does every 1024th line |
| GL_LOD_VLOW | Level Of Detail | 0x00A0 | Sets the LOD to Very Low, Does every 2048th line |
| GL_LUM_YUV | Color Space | 0x0001 | Configures the Color Space Equation to Digital YUV |
| GL_LUM_HID | Color Space | 0x0002 | Configures the Color Space Equation to High Definition |
| GL_LUM_STD | Color Space | 0x0003 | Configures the Color Space Equation to Standard Definition |