# OSMesa Off-screen Rendering

Mesa’s off-screen interface is used for rendering into user-allocated memory without any sort of window system or operating system dependencies. That is, the GL\_FRONT colorbuffer is actually a buffer in main memory, rather than a window on your display.

The OSMesa API provides three basic functions for making off-screen renderings: OSMesaCreateContext(), OSMesaMakeCurrent(), and OSMesaDestroyContext(). See the Mesa/include/GL/osmesa.h header for more information about the API functions.

The OSMesa interface may be used with the gallium software renderers:

1. llvmpipe - this is the high-performance Gallium LLVM driver
2. softpipe - this it the reference Gallium software driver

There are several examples of OSMesa in the mesa/demos repository.

## Building OSMesa

Configure and build Mesa with something like:

meson builddir -Dosmesa=true -Dgallium-drivers=swrast -Ddri-drivers=[] -Dvulkan-drivers=[] -Dprefix=$PWD/builddir/install

ninja -C builddir install

Make sure you have LLVM installed first if you want to use the llvmpipe driver.

When the build is complete you should find:编译成功的Mesa动态链接库：

$PWD/builddir/install/lib/libOSMesa.so

Set your LD\_LIBRARY\_PATH to point to $PWD/builddir/install to use the libraries

When you link your application, link with -lOSMesa

# Paraview的使用EGL实现Off-screen Rendering

ParaView can run on a supercomputer with thousands of nodes to provide visualization and analysis of very large datasets. In this configuration, the same version of the ParaView analysis pipeline runs on each node to process a piece of the data, the results are rendered in software using [Off-Screen Mesa](http://www.mesa3d.org/osmesa.html) and composited into a final image which is send to the ParaView client for display.

Software rendering is used because, until recently, supercomputer nodes did not provide graphic cards as they were used mainly for computation. This is beginning to change with the release of new GPU Accelerators cards, such as NVIDIA Tesla, which can be used for both computation and off-screen rendering.

The Native Platform Interface (EGL) provides means to render to a native windowing system, such as Android, X Windows or Microsoft Windows, or to an off-screen buffer (without a need of a windowing system). For rendering API, one can choose OpenGL ES, OpenVG or, starting with EGL version 1.4, full OpenGL.

We enable the VTK and the ParaView server (**pvserver**) to render to an EGL off-screen buffer. Through this work we allow server-side hardware-accelerated rendering without the need to install a windowing system.

## Configuration parameters

To compile VTK or ParaView for off-screen rendering through EGL you will need:

1. A graphics card driver that supports OpenGL rendering through EGL (full OpenGL rendering is supported only in EGL version 1.4 or later). We have tested our code with the [NVIDIA driver version 355.11](http://www.nvidia.com/Download/driverResults.aspx/90393/en-us).
2. You might need the EGL headers as they did not come with the Nvidia driver used in our tests. You can download them from [Khronos EGL Registry](https://www.khronos.org/registry/egl/).
3. Set VTK advanced configuration option **VTK\_USE\_OFFSCREEN\_EGL**

You’ll get a configuration error if any of the windowing systems is enabled: **VTK\_USE\_X** or **VTK\_USE\_COCOA** so you’ll have to disable your windowing system. You’ll also get an error if you are on **WIN32**​, **ANDROID** or **APPLE\_IOS**.

If you have several graphics cards on you system you may need to set the index of the graphics card you want to use, if that is different than the default card chosen by the driver. You can do that if your driver supports **EGL\_EXT\_platform\_device** and **EGL\_EXT\_device\_base** extensions.

You can set the default graphics card used by the render window in VTK by setting the advanced configuration option **VTK\_EGL\_DEVICE\_INDEX** to an integer such as **0** or **1** for two cards installed on a system. By default, this variable is set to **0** which means that the default graphics card is used. We are investigating using a more user friendly mechanism such as the name of the graphics card. We note that the index of the graphics card you need to pass is the same as the index of the card returned by the following command **nvidia-smi.**

## Runtime parameters

For a system with more then one graphics card installed, you can choose the graphics card used for rendering at runtime, in case it is different that the card setup at configuration time.

## VTK

If you want to change the graphics card set through the configuration process, you can call **vtkRenderWindow::GetNumberOfDevices()** to query the number of devices available on a system and **vtkRenderWindow:: SetDeviceIndex(deviceIndex)** to set the device you want to be used for rendering.

## ParaView

To start **pvserver**​ with rendering set on a graphics card different than the card set through the configuration process, you have to pass the following command line parameter:

**–egl-device-index=<device\_index>**, where <device\_index> is the graphics card index.

To check if you are rendering to the correct graphics card in ParaView you can use Help, About, Connection Information, OpenGL Renderer.

## Troubleshooting

1. Make sure that **EGL\_INCLUDE\_DIR**, **EGL\_LIBRARY**, **EGL\_gldispatch\_LIBRARY**, **EGL\_opengl\_LIBRARY** point to valid headers and libraries. On Ubuntu 16.04 with NVidia driver version 361.42 the libraries are: /usr/lib/nvidia-361/libEGL.so, /usr/lib/nvidia-361/libGLdispatch.so.0 and /usr/lib/nvidia-361/libOpenGL.so.
2. Pass **–disable-xdisplay-test** to **pvserver** if this option exists. We have seen a case when this test creates problems with the EGL rendering

We hope you enjoy this new feature. It is available in the VTK and ParaView git repositories.

# ParaView and Offscreen Rendering

ParaView is often used to render visualization results. While in most cases, the results are presented to the users on their screens or monitors, there are cases when this is not required. For example, if running a batch script to generate several images, one may not care to see windows flashing in front of them while the results are being generated. Another use-case is when using a **pvserver** or **pvrenderserver** to render images remotely and having them sent back to the client for viewing. The server-side windows are do not serve any useful purpose in that mode. In other cases, showing a window may not be possible at all, for example on certain Linux clusters, one may not have access to an X server at all, in which case on-screen rendering is just not possible (this is also referred to as headless operating mode).

This page documents the onscreen, offscreen, and headless rendering support in ParaView.

## 术语

A brief explanation of the terms:

1. **Desktop Qt client**: This refers to the Qt-based GUI. This is launched using the **paraview** executable.
2. **Command line executables**: This refers to all other ParaView executables, e.g. **pvserver**, **pvdataserver**, **pvrenderserver**, **pvpython**, and **pvbatch**.
3. **Onscreen**: Onscreen refers to the case where the rendering results are presented to the users in a viewable window. On Linux, this invariably requires a running and accessible X server. The desktop Qt client can only operate in on-screen mode and hence needs an accessible X server.
4. **Offscreen**: Offscreen simply means that the rendering results are not presented to the user in a window. On Linux, this does not automatically imply that an accessible X server is not needed. X server may be needed, unless ParaView was built with an OpenGL implementation that allows for headless operation. This mode is not applicable to the desktop Qt client and is only supported by the command line executables.
5. **Headless**: Headless rendering automatically implies offscreen rendering. In addition, on Linux, it implies that the rendering does not require an accessible X server nor will it make any X calls.

**Onscreen** and **offscreen** support is built by default. Thus ParaView binaries available from paraview.org support these modes. **Headless** support requires special builds of ParaView with runtime capabilities that are not widely available yet. Hence currently (as of ParaView 5.4) one has to build ParaView from source with special build flags to enable headless support.

## 实施OpenGL

ParaView uses OpenGL for rendering. OpenGL is an API specification for 2D/3D rendering. Many vendors provide implementations of this API, including those that build GPUs.

For sake of simplicity, let's classify OpenGL implementations as **hardware (H/W)** and **software (S/W)**. **H/W** includes OpenGL implementations provided by NVIDIA, ATI, Intel, Apple and others which typically use the system hardware infrastructure for rendering. The runtime libraries needed for these are available on the system. **S/W** currently includes Mesa3D – a software implementation of the OpenGL standard. Despite the names, H/W doesn't necessarily imply use of GPUs, nor does S/W imply exclusion of GPUs. Nonetheless we use this naming scheme as it has been prevalent.

APIs for Headless Support

Traditionally, OpenGL implementations are coupled with the window system to provide an OpenGL context. Thus, they are designed for non-headless operation. When it comes to headless operation, there are alternative APIs that an application can use to create the OpenGL context that avoid this dependency on the window system (or X server for the sake of this discussion).

Currently, ParaView supports two distinct APIs that are available for headless operation: **EGL** and **OSMesa** (also called **Offscreen Mesa**). It must be noted that headless support is a rapidly evolving area and changes are expected in coming months. Modern H/W OpenGL implementations support EGL while S/W (or Mesa) supports OSMesa. One has to build ParaView with specific CMake flags changed to enable either of these APIs. Which headless API you choose in your build depends on which OpenGL implementation you plan to use.

## 编译Paraview

Before we look at the various ways you can build and use ParaView, let's summarize relevant CMake options available:

* VTK\_USE\_X: When ON, implies that ParaView can link against X libraries. This allows ParaView executables to create on-screen windows, if needed.

When VTK\_USE\_X is ON, these variables must be specified:

* + OPENGL\_INCLUDE\_DIR: Path to directory containing GL/gl.h.
  + OPENGL\_gl\_LIBRARY: Path to libGL.so.
  + OpengL\_glu\_LIBRARY: not needed for ParaView; leave empty.
  + OPENGL\_xmesa\_INCLUDE\_DIR: not needed for ParaView; leave empty.
* VTK\_OPENGL\_HAS\_OSMESA: When ON, implies that ParaView can use OSMesa to support headless modes of operation. When VTK\_OPENGL\_HAS\_OSMESA is ON, these variables must be specified:
  + OSMESA\_INCLUDE\_DIR: Path to containing GL/osmesa.h.
  + OSMESA\_LIBRARY: Path to libOSMesa.so
* VTK\_OPENGL\_HAS\_EGL: When ON, implies that ParaView can use EGL to support headless modes of operation.

When VTK\_OPENGL\_HAS\_EGL is ON, these variables must be specified:

* + EGL\_INCLUDE\_DIR: Path to directory containing GL/egl.h.
  + EGL\_LIBRARY: Path to libEGL.so.
  + EGL\_opengl\_LIBRARY: Path to libOpenGL.so.
* PARAVIEW\_USE\_QT  indicates if the desktop Qt client should be built.

All combinations of above options can be turned on or off independently except that presently VTK\_OPENGL\_HAS\_EGL and VTK\_OPENGL\_HAS\_OSMESA are mutually exclusive i.e. only one of the two can be ON at the same time. This is because the current version of Mesa (17.1.5) doesn't support EGL for OpenGL, it's only supported for OpenGL-ES. EGL与Mesa的编译支持互斥！

还需要注意的有：

* If VTK\_OPENGL\_HAS\_EGL or VTK\_OPENGL\_HAS\_OSMESA is ON, the build supports headless rendering, otherwise VTK\_USE\_X must be ON and the build does not support headless, but can still support offscreen rendering.
* If VTK\_USE\_X is OFF, then either VTK\_OPENGL\_HAS\_OSMESA or VTK\_OPENGL\_HAS\_EGL must be ON. Then the build does not support onscreen rendering, but only headless rendering.
* If PARAVIEW\_USE\_QT is ON and VTK\_USE\_X is ON, while ParaView command line tools won't link against or use X calls, Qt will and hence an accessible X server is still needed to run the desktop client.
* If VTK\_OPENGL\_HAS\_OSMESA is ON, and VTK\_USE\_X is ON, then all the OpenGL and OSMesa variables should point to the Mesa libraries.
* Likewise, if VTK\_OPENGL\_HAS\_EGL is ON and VTK\_USE\_X is ON, then all the OpenGL and EGL variables should point to the system libraries providing both, typically the NVidia libraries.

## 默认的渲染模式

Since now it's possible to build ParaView with onscreen and headless support simultaneously, which type of render window the ParaView executable creates by default also needs some explanation.

* The ParaView desktop Qt client always creates an onscreen window using GLX calls via Qt.
* pvserver, pvrenderserver and pvbatch always create an offscreen render window. If built with headless support, it will be an offscreen-headless window. There are a few exceptions where it will create an onscreen window instead:
  + if running in tile-display mode, i.e. -tdx or -tdy command line options are passed
  + if running in immersive mode e.g. CAVE
  + if PV\_DEBUG\_REMOTE\_RENDERING environment is set
  + if --force-onscreen-rendering command line option is passed.
* pvpython always creates an onscreen render window if built with onscreen support. The following are the exceptions when it creates offscreen (or headless if supported) render windows.
  + if --force-offscreen-rendering command line option is passed.

--use-offscreen-rendering command line option supported by ParaView 5.4 and earlier has now been deprecated and is interpreted as --force-offscreen-rendering.