Mesa Documentation

Release 12.0

Brian Paul et al.

August 28, 2016

CHAPTER 1

Introduction

Mesa is an open-source implementation of the OpenGL specification - a system for rendering interactive 3D graphics.

A variety of device drivers allows Mesa to be used in many different environments ranging from software emulation to complete hardware acceleration for modern GPUs.

Mesa ties into several other open-source projects: the Direct Rendering Infrastructure and X.org to provide OpenGL support to users of X on Linux, FreeBSD and other operating systems.

Project History

The Mesa project was originally started by Brian Paul. Here's a short history of the project.

August, 1993: I begin working on Mesa in my spare time. The project has no name at that point. I was simply interested in writing a simple 3D graphics library that used the then-new OpenGL API. I was partially inspired by the *VOGL* library which emulated a subset of IRIS GL. I had been programming with IRIS GL since 1991.

November 1994: I contact SGI to ask permission to distribute my OpenGL-like graphics library on the internet. SGI was generally receptive to the idea and after negotiations with SGI's legal department, I get permission to release it.

February 1995: Mesa 1.0 is released on the internet. I expected that a few people would be interested in it, but not thousands. I was soon receiving patches, new features and thank-you notes on a daily basis. That encouraged me to continue working on Mesa. The name Mesa just popped into my head one day. SGI had asked me not to use the terms *"Open"* or *"GL"* in the project name and I didn't want to make up a new acronym. Later, I heard of the Mesa programming language and the Mesa spreadsheet for NeXTStep.

In the early days, OpenGL wasn't available on too many systems. It even took a while for SGI to support it across their product line. Mesa filled a big hole during that time. For a lot of people, Mesa was their first introduction to OpenGL. I think SGI recognized that Mesa actually helped to promote the OpenGL API, so they didn't feel threatened by the project.

1995-1996: I continue working on Mesa both during my spare time and during my work hours at the Space Science and Engineering Center at the University of Wisconsin in Madison. My supervisor, Bill Hibbard, lets me do this because Mesa is now being using for the Vis5D project.

October 1996: Mesa 2.0 is released. It implements the OpenGL 1.1 specification.

March 1997: Mesa 2.2 is released. It supports the new 3dfx Voodoo graphics card via the Glide library. It's the first really popular hardware OpenGL implementation for Linux.

September 1998: Mesa 3.0 is released. It's the first publicly-available implementation of the OpenGL 1.2 API.

March 1999: I attend my first OpenGL ARB meeting. I contribute to the development of several official OpenGL extensions over the years.

September 1999: I'm hired by Precision Insight, Inc. Mesa is a key component of 3D hardware acceleration in the new DRI project for XFree86. Drivers for 3dfx, 3dLabs, Intel, Matrox and ATI hardware soon follow.

October 2001: Mesa 4.0 is released. It implements the OpenGL 1.3 specification.

November 2001: I cofounded Tungsten Graphics, Inc. with Keith Whitwell, Jens Owen, David Dawes and Frank LaMonica. Tungsten Graphics was acquired by VMware in December 2008.

November 2002: Mesa 5.0 is released. It implements the OpenGL 1.4 specification.

January 2003: Mesa 6.0 is released. It implements the OpenGL 1.5 specification as well as the GL_ARB_vertex_program and GL_ARB_fragment_program extensions.

June 2007: Mesa 7.0 is released, implementing the OpenGL 2.1 specification and OpenGL Shading Language.

2008: Keith Whitwell and other Tungsten Graphics employees develop Gallium - a new GPU abstraction layer. The latest Mesa drivers are based on Gallium and other APIs such as OpenVG are implemented on top of Gallium.

February 2012: Mesa 8.0 is released, implementing the OpenGL 3.0 specification and version 1.30 of the OpenGL Shading Language.

Ongoing: Mesa is the OpenGL implementation for several types of hardware made by Intel, AMD and NVIDIA, plus the VMware virtual GPU. There's also several software-based renderers: swrast (the legacy Mesa rasterizer), softpipe (a gallium reference driver) and llvmpipe (LLVM/JIT-based high-speed rasterizer). Work continues on the drivers and core Mesa to implement newer versions of the OpenGL specification.

Major Versions

This is a summary of the major versions of Mesa. Mesa's major version number has been incremented whenever a new version of the OpenGL specification is implemented.

Version 9.x features

Version 9.x of Mesa implements the OpenGL 3.1 API. While the driver for Intel Sandy Bridge and Ivy Bridge is the only driver to support OpenGL 3.1, many developers across the open-source community contributed features required for OpenGL 3.1. The primary features added since the Mesa 8.0 release are GL_ARB_texture_buffer_object and GL_ARB_uniform_buffer_object.

Version 8.x features

Version 8.x of Mesa implements the OpenGL 3.0 API. The developers at Intel deserve a lot of credit for implementing most of the OpenGL 3.0 features in core Mesa, the GLSL compiler as well as the i965 driver.

Version 7.x features

Version 7.x of Mesa implements the OpenGL 2.1 API. The main feature of OpenGL 2.x is the OpenGL Shading Language.

Version 6.x features

Version 6.x of Mesa implements the OpenGL 1.5 API with the following extensions incorporated as standard features:

- GL_ARB_occlusion_query

- GL_ARB_vertex_buffer_object
- GL_EXT_shadow_funcs

Also note that several OpenGL tokens were renamed in OpenGL 1.5 for the sake of consistency. The old tokens are still available.

New Token	Old Token
GL_FOG_COORD_SRC	GL_FOG_COORDINATE_SOURCE
GL_FOG_COORD	GL_FOG_COORDINATE
GL_CURRENT_FOG_COORD	GL_CURRENT_FOG_COORDINATE
GL_FOG_COORD_ARRAY_TYPE	GL_FOG_COORDINATE_ARRAY_TYPE
GL_FOG_COORD_ARRAY_STRIDE	GL_FOG_COORDINATE_ARRAY_STRIDE
GL_FOG_COORD_ARRAY_POINTER	GL_FOG_COORDINATE_ARRAY_POINTER
GL_FOG_COORD_ARRAY	GL_FOG_COORDINATE_ARRAY
GL_SRC0_RGB	GL_SOURCE0_RGB
GL_SRC1_RGB	GL_SOURCE1_RGB
GL_SRC2_RGB	GL_SOURCE2_RGB
GL_SRC0_ALPHA	GL_SOURCE0_ALPHA
GL_SRC1_ALPHA	GL_SOURCE1_ALPHA
GL_SRC2_ALPHA	GL_SOURCE2_ALPHA

See the OpenGL specification for more details.

Version 5.x features

Version 5.x of Mesa implements the OpenGL 1.4 API with the following extensions incorporated as standard features:

- GL_ARB_depth_texture
- GL_ARB_shadow
- GL_ARB_texture_env_crossbar
- GL_ARB_texture_mirror_repeat
- GL_ARB_window_pos
- GL_EXT_blend_color
- GL_EXT_blend_func_separate
- GL_EXT_blend_logic_op
- GL_EXT_blend_minmax
- GL_EXT_blend_subtract
- GL_EXT_fog_coord
- GL_EXT_multi_draw_arrays
- GL_EXT_point_parameters
- GL_EXT_secondary_color
- GL_EXT_stencil_wrap
- GL_EXT_texture_lod_bias (plus, a per-texture LOD bias parameter)
- GL_SGIS_generate_mipmap

Version 4.x features

Version 4.x of Mesa implements the OpenGL 1.3 API with the following extensions incorporated as standard features:

- GL_ARB_multisample
- GL_ARB_multitexture
- GL_ARB_texture_border_clamp
- GL_ARB_texture_compression
- GL_ARB_texture_cube_map
- GL_ARB_texture_env_add
- GL_ARB_texture_env_combine
- GL_ARB_texture_env_dot3
- GL_ARB_transpose_matrix

Version 3.x features

Version 3.x of Mesa implements the OpenGL 1.2 API with the following features:

- BGR, BGRA and packed pixel formats
- New texture border clamp mode
- glDrawRangeElements()
- standard 3-D texturing
- advanced MIPMAP control
- separate specular color interpolation

Version 2.x features

Version 2.x of Mesa implements the OpenGL 1.1 API with the following features.

- Texture mapping:
  - glAreTexturesResident
  - glBindTexture
  - glCopyTexImage1D
  - glCopyTexImage2D
  - glCopyTexSubImage1D
  - glCopyTexSubImage2D
  - glDeleteTextures
  - glGenTextures
  - glIsTexture
  - glPrioritizeTextures
  - glTexSubImage1D

  - glTexSubImage2D
- Vertex Arrays:
  - glArrayElement
  - glColorPointer
  - glDrawElements
  - glEdgeFlagPointer
  - glIndexPointer
  - glInterleavedArrays
  - glNormalPointer
  - glTexCoordPointer
  - glVertexPointer
- Client state management:
  - glDisableClientState
  - glEnableClientState
  - glPopClientAttrib
  - glPushClientAttrib
- Misc:
  - glGetPointer
  - glIndexub
  - glIndexubv
  - glPolygonOffset

CHAPTER 2

Developers

Both professional and volunteer developers contribute to Mesa.

VMware employs several of the main Mesa developers including Brian Paul and Keith Whitwell.

In the past, Tungsten Graphics contracts implemented many Mesa features including:

- DRI drivers for Intel i965, i945, i915 and other chips
- Advanced memory manager and framebuffer object support
- Shading language compiler and OpenGL 2.0 support
- MiniGLX environment

Other companies including Intel and RedHat also actively contribute to the project. Intel has recently contributed the new GLSL compiler in Mesa 7.9.

LunarG can be contacted for custom Mesa / 3D graphics development.

Volunteers have made significant contributions to all parts of Mesa, including complete device drivers.

CHAPTER 3

Supported Systems and Drivers

Mesa is primarily developed and used on Linux systems. But there's also support for Windows, other flavors of Unix and other systems such as Haiku. We're actively developing and maintaining several hardware and software drivers.

The primary API is OpenGL but there's also support for OpenGL ES 1, ES2 and ES 3, OpenVG, OpenCL, VDPAU, XvMC and the EGL interface.

Hardware drivers include:

- Intel i965, i945, i915. See Intel's website
- AMD Radeon series. See RadeonFeature
- NVIDIA GPUs. See Nouveau Wiki
- VMware virtual GPU

Software drivers include:

- llvmpipe - uses LLVM for x86 JIT code generation and is multi-threaded
- softpipe - a reference Gallium driver
- swrast - the legacy/original Mesa software rasterizer

Additional driver information:

- DRI hardware drivers for the X Window System
- Xlib / swrast driver for the X Window System and Unix-like operating systems
- Microsoft Windows
- VMware guest OS driver

Deprecated Systems and Drivers

In the past there were other drivers for older GPUs and operating systems. These have been removed from the Mesa source tree and distribution. If anyone's interested though, the code can be found in the git repo. The list includes:

- 3dfx/glide
- Matrox
- ATI R128
- Savage
- VIA Unichrome

- SIS
- 3Dlabs gamma
- DOS
- fbdev
- DEC/VMS

CHAPTER 4

Disclaimer

Mesa is a 3-D graphics library with an API which is very similar to that of OpenGL.* To the extent that Mesa utilizes the OpenGL command syntax or state machine, it is being used with authorization from Silicon Graphics, Inc.(SGI). However, the author does not possess an OpenGL license from SGI, and makes no claim that Mesa is in any way a compatible replacement for OpenGL or associated with SGI. Those who want a licensed implementation of OpenGL should contact a licensed vendor.

Please do not refer to the library as *MesaGL* (for legal reasons). It's just *Mesa* or *The Mesa 3-D graphics library*.

* OpenGL is a trademark of Silicon Graphics Incorporated.

License / Copyright Information

The Mesa distribution consists of several components. Different copyrights and licenses apply to different components. For example, the GLX client code uses the SGI Free Software License B, and some of the Mesa device drivers are copyrighted by their authors. See below for a list of Mesa's main components and the license for each.

The core Mesa library is licensed according to the terms of the MIT license. This allows integration with the XFree86, Xorg and DRI projects.

The default Mesa license is as follows:

```
Copyright (C) 1999-2007  Brian Paul   All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Attention, Contributors

When contributing to the Mesa project you must agree to the licensing terms of the component to which you're contributing. The following section lists the primary components of the Mesa distribution and their respective licenses.

Mesa Component Licenses

Component	Location	License
Main Mesa code	src/mesa/	MIT
Device drivers	src/mesa/drivers/*	MIT, generally
Gallium code	src/gallium/	MIT
Ext headers	include/GL/glext.h include/GL/glxext.h	Khronos
GLX client code	src/glx/	SGI Free Software License B
C11 thread emulation	include/c11/threads*.h	Boost (permissive)

In general, consult the source files for license terms.

CHAPTER 5

Mesa Frequently Asked Questions

Last updated: 9 October 2012 .. rubric:: Index

name index

1. High-level Questions and Answers 2. Compilation and Installation Problems 3. Runtime / Rendering Problems 4. Developer Questions .. rubric:: 1. High-level Questions and Answers

name part1

1.1 What is Mesa?

Mesa is an open-source implementation of the OpenGL specification. OpenGL is a programming library for writing interactive 3D applications. See the OpenGL website for more information.

Mesa 9.x supports the OpenGL 3.1 specification.

1.2 Does Mesa support/use graphics hardware?

Yes. Specifically, Mesa serves as the OpenGL core for the open-source DRI drivers for X.org.

- See the DRI website for more information.
- See 01.org for more information about Intel drivers.
- See nouveau.freedesktop.org for more information about Nouveau drivers.
- See www.x.org/wiki/RadeonFeature for more information about Radeon drivers.

1.3 What purpose does Mesa serve today?

Hardware-accelerated OpenGL implementations are available for most popular operating systems today. Still, Mesa serves at least these purposes:

- Mesa is used as the core of the open-source X.org DRI hardware drivers.
- Mesa is quite portable and allows OpenGL to be used on systems that have no other OpenGL solution.
- Software rendering with Mesa serves as a reference for validating the hardware drivers.
- A software implementation of OpenGL is useful for experimentation, such as testing new rendering techniques.
- Mesa can render images with deep color channels: 16-bit integer and 32-bit floating point color channels are supported. This capability is only now appearing in hardware.

- Mesa's internal limits (max lights, clip planes, texture size, etc) can be changed for special needs (hardware limits are hard to overcome).

1.4 What's the difference between "Stand-Alone" Mesa and the DRI drivers?

Stand-alone Mesa is the original incarnation of Mesa. On systems running the X Window System it does all its rendering through the Xlib API:

- The GLX API is supported, but it's really just an emulation of the real thing.
- The GLX wire protocol is not supported and there's no OpenGL extension loaded by the X server.
- There is no hardware acceleration.
- The OpenGL library, libGL.so, contains everything (the programming API, the GLX functions and all the rendering code).

Alternately, Mesa acts as the core for a number of OpenGL hardware drivers within the DRI (Direct Rendering Infrastructure):

- The libGL.so library provides the GL and GLX API functions, a GLX protocol encoder, and a device driver loader.
- The device driver modules (such as r200_dri.so) contain a built-in copy of the core Mesa code.
- The X server loads the GLX module. The GLX module decodes incoming GLX protocol and dispatches the commands to a rendering module. For the DRI, this module is basically a software Mesa renderer.

1.5 How do I upgrade my DRI installation to use a new Mesa release?

This wasn't easy in the past. Now, the DRI drivers are included in the Mesa tree and can be compiled separately from the X server. Just follow the Mesa compilation instructions.

1.6 Are there other open-source implementations of OpenGL?

Yes, SGI's OpenGL Sample Implementation (SI) is available. The SI was written during the time that OpenGL was originally designed. Unfortunately, development of the SI has stagnated. Mesa is much more up to date with modern features and extensions.

Vincent is an open-source implementation of OpenGL ES for mobile devices.

miniGL is a subset of OpenGL for PalmOS devices.

TinyGL is a subset of OpenGL.

SoftGL is an OpenGL subset for mobile devices.

Chromium isn't a conventional OpenGL implementation (it's layered upon OpenGL), but it does export the OpenGL API. It allows tiled rendering, sort-last rendering, etc.

ClosedGL is an OpenGL subset library for TI graphing calculators.

There may be other open OpenGL implementations, but Mesa is the most popular and feature-complete.

2. Compilation and Installation Problems

2.1 What's the easiest way to install Mesa?

If you're using a Linux-based system, your distro CD most likely already has Mesa packages (like RPM or DEB) which you can easily install.

2.2 I get undefined symbols such as bgnpolygon, v3f, etc...

You're application is written in IRIS GL, not OpenGL. IRIS GL was the predecessor to OpenGL and is a different thing (almost) entirely. Mesa's not the solution.

2.3 Where is the GLUT library?

GLUT (OpenGL Utility Toolkit) is no longer in the separate MesaGLUT-x.y.z.tar.gz file. If you don't already have GLUT installed, you should grab freeglut.

2.4 Where is the GLw library?

GLw (OpenGL widget library) is now available from a separate git repository. Unless you're using very old Xt/Motif applications with OpenGL, you shouldn't need it.

2.5 What's the proper place for the libraries and headers?

On Linux-based systems you'll want to follow the Linux ABI standard. Basically you'll want the following:

- /usr/include/GL/gl.h - the main OpenGL header
- /usr/include/GL/glu.h - the OpenGL GLU (utility) header
- /usr/include/GL/glx.h - the OpenGL GLX header
- /usr/include/GL/glext.h - the OpenGL extensions header
- /usr/include/GL/glxext.h - the OpenGL GLX extensions header
- /usr/include/GL/osmesa.h - the Mesa off-screen rendering header
- /usr/lib/libGL.so - a symlink to libGL.so.1
- /usr/lib/libGL.so.1 - a symlink to libGL.so.1.xyz
- /usr/lib/libGL.so.xyz - the actual OpenGL/Mesa library. xyz denotes the Mesa version number.

When configuring Mesa, there are three autoconf options that affect the install location that you should take care with: `--prefix`, `--libdir`, and `--with-dri-driverdir`. To install Mesa into the system location where it will be available for all programs to use, set `--prefix=/usr`. Set `--libdir` to where your Linux distribution installs system libraries, usually either `/usr/lib` or `/usr/lib64`. Set `--with-dri-driverdir` to the directory where your Linux distribution installs DRI drivers. To find your system's DRI driver directory, try executing `find /usr -type d -name dri`. For example, if the `find` command listed `/usr/lib64/dri`, then set `--with-dri-driverdir=/usr/lib64/dri`.

After determining the correct values for the install location, configure Mesa with `./configure --prefix=/usr --libdir=xxx --with-dri-driverdir=xxx` and then install with `sudo make install`.

3. Runtime / Rendering Problems

3.1 Rendering is slow / why isn't my graphics hardware being used?

If Mesa can't use its hardware accelerated drivers it falls back on one of its software renderers. (eg. classic swrast, softpipe or llvmpipe)

You can run the `glxinfo` program to learn about your OpenGL library. Look for the `OpenGL vendor` and `OpenGL renderer` values. That will identify who's OpenGL library with which driver you're using and what sort of hardware it has detected.

If you're using a hardware accelerated driver you want `direct rendering: Yes`.

If your DRI-based driver isn't working, go to the DRI website for trouble-shooting information.

3.2 I'm seeing errors in depth (Z) buffering. Why?

Make sure the ratio of the far to near clipping planes isn't too great. Look here for details.

Mesa uses a 16-bit depth buffer by default which is smaller and faster to clear than a 32-bit buffer but not as accurate. If you need a deeper you can modify the parameters to '' glXChooseVisual'' in your code.

3.3 Why Isn't depth buffering working at all?

Be sure you're requesting a depth buffered-visual. If you set the MESA_DEBUG environment variable it will warn you about trying to enable depth testing when you don't have a depth buffer.

Specifically, make sure `glutInitDisplayMode` is being called with `GLUT_DEPTH` or `glXChooseVisual` is being called with a non-zero value for GLX_DEPTH_SIZE.

This discussion applies to stencil buffers, accumulation buffers and alpha channels too.

3.4 Why does glGetString() always return NULL?

Be sure you have an active/current OpenGL rendering context before calling glGetString.

3.5 GL_POINTS and GL_LINES don't touch the right pixels

If you're trying to draw a filled region by using GL_POINTS or GL_LINES and seeing holes or gaps it's because of a float-to-int rounding problem. But this is not a bug. See Appendix H of the OpenGL Programming Guide - "OpenGL Correctness Tips". Basically, applying a translation of (0.375, 0.375, 0.0) to your coordinates will fix the problem.

4. Developer Questions

4.1 How can I contribute?

First, join the mesa-dev mailing list. That's where Mesa development is discussed.

The OpenGL Specification is the bible for OpenGL implementation work. You should read it.

Most of the Mesa development work involves implementing new OpenGL extensions, writing hardware drivers (for the DRI), and code optimization.

4.2 How do I write a new device driver?

Unfortunately, writing a device driver isn't easy. It requires detailed understanding of OpenGL, the Mesa code, and your target hardware/operating system. 3D graphics are not simple.

The best way to get started is to use an existing driver as your starting point. For a classic hardware driver, the i965 driver is a good example. For a Gallium3D hardware driver, the r300g, r600g and the i915g are good examples.

The DRI website has more information about writing hardware drivers. The process isn't well document because the Mesa driver interface changes over time, and we seldom have spare time for writing documentation. That being said, many people have managed to figure out the process.

Joining the appropriate mailing lists and asking questions (and searching the archives) is a good way to get information.

4.3 Why isn't GL_EXT_texture_compression_s3tc implemented in Mesa?

The specification for the extension indicates that there are intellectual property (IP) and/or patent issues to be dealt with.

We've been unsuccessful in getting a response from S3 (or whoever owns the IP nowadays) to indicate whether or not an open source project can implement the extension (specifically the compression/decompression algorithms).

In the mean time, a 3rd party plug-in library is available.

CHAPTER 6

Release Notes

The release notes summarize what's new or changed in each Mesa release.

- 12.0.1 release notes
- 12.0.0 release notes
- 11.2.2 release notes
- 11.1.4 release notes
- 11.2.1 release notes
- 11.1.3 release notes
- 11.2.0 release notes
- 11.1.2 release notes
- 11.0.9 release notes
- 11.1.1 release notes
- 11.0.8 release notes
- 11.1.0 release notes
- 11.0.7 release notes
- 11.0.6 release notes
- 11.0.5 release notes
- 11.0.4 release notes
- 11.0.3 release notes
- 10.6.9 release notes
- 11.0.2 release notes
- 11.0.1 release notes
- 10.6.8 release notes
- 11.0.0 release notes
- 10.6.7 release notes
- 10.6.6 release notes
- 10.6.5 release notes

- 10.6.4 release notes
- 10.6.3 release notes
- 10.6.2 release notes
- 10.5.9 release notes
- 10.6.1 release notes
- 10.5.8 release notes
- 10.6.0 release notes
- 10.5.7 release notes
- 10.5.6 release notes
- 10.5.5 release notes
- 10.5.4 release notes
- 10.5.3 release notes
- 10.5.2 release notes
- 10.4.7 release notes
- 10.5.1 release notes
- 10.5.0 release notes
- 10.4.6 release notes
- 10.4.5 release notes
- 10.4.4 release notes
- 10.4.3 release notes
- 10.4.2 release notes
- 10.3.7 release notes
- 10.4.1 release notes
- 10.3.6 release notes
- 10.4 release notes
- 10.3.5 release notes
- 10.3.4 release notes
- 10.3.3 release notes
- 10.3.2 release notes
- 10.3.1 release notes
- 10.2.9 release notes
- 10.3 release notes
- 10.2.8 release notes
- 10.2.7 release notes
- 10.2.6 release notes
- 10.2.5 release notes

- 10.2.4 release notes
- 10.2.3 release notes
- 10.2.2 release notes
- 10.2.1 release notes
- 10.2 release notes
- 10.1.6 release notes
- 10.1.5 release notes
- 10.1.4 release notes
- 10.1.3 release notes
- 10.1.2 release notes
- 10.1.1 release notes
- 10.1 release notes
- 10.0.5 release notes
- 10.0.4 release notes
- 10.0.3 release notes
- 10.0.2 release notes
- 10.0.1 release notes
- 10.0 release notes
- 9.2.5 release notes
- 9.2.4 release notes
- 9.2.3 release notes
- 9.2.2 release notes
- 9.2.1 release notes
- 9.2 release notes
- 9.1.7 release notes
- 9.1.6 release notes
- 9.1.5 release notes
- 9.1.4 release notes
- 9.1.3 release notes
- 9.1.2 release notes
- 9.1.1 release notes
- 9.1 release notes
- 9.0.3 release notes
- 9.0.2 release notes
- 9.0.1 release notes
- 9.0 release notes

- 8.0.5 release notes
- 8.0.4 release notes
- 8.0.3 release notes
- 8.0.2 release notes
- 8.0.1 release notes
- 8.0 release notes
- 7.11.2 release notes
- 7.11.1 release notes
- 7.11 release notes
- 7.10.3 release notes
- 7.10.2 release notes
- 7.10.1 release notes
- 7.10 release notes
- 7.9.2 release notes
- 7.9.1 release notes
- 7.9 release notes
- 7.8.3 release notes
- 7.8.2 release notes
- 7.8.1 release notes
- 7.8 release notes
- 7.7.1 release notes
- 7.7 release notes
- 7.6.1 release notes
- 7.6 release notes
- 7.5.2 release notes
- 7.5.1 release notes
- 7.5 release notes
- 7.4.4 release notes
- 7.4.3 release notes
- 7.4.2 release notes
- 7.4.1 release notes
- 7.4 release notes
- 7.3 release notes
- 7.2 release notes
- 7.1 release notes
- 7.0.4 release notes

- 7.0.3 release notes
- 7.0.2 release notes
- 7.0.1 release notes
- 7.0 release notes
- 6.5.3 release notes
- 6.5.2 release notes
- 6.5.1 release notes
- 6.5 release notes
- 6.4.2 release notes
- 6.4.1 release notes
- 6.4 release notes

Versions of Mesa prior to 6.4 are summarized in the versions file and the following release notes.

- 6.3.2 release notes
- 6.3.1 release notes
- 6.3 release notes
- 6.2.1 release notes
- 6.2 release notes
- 6.1 release notes
- 6.0.1 release notes
- 6.0 release notes
- 5.1 release notes
- 5.0.2 release notes
- 5.0.1 release notes
- 5.0 release notes
- 4.1 release notes
- 4.0.3 release notes
- 4.0.2 release notes
- 4.0.1 release notes
- 4.0 release notes
- 3.5 release notes
- 3.4.2 release notes
- 3.4.1 release notes
- 3.4 release notes
- 3.3 release notes
- 3.2.1 release notes
- 3.2 release notes

- 3.1 release notes

CHAPTER 7

Acknowledgements

The following individuals and groups are to be acknowledged for their contributions to Mesa over the years. This list is far from complete and somewhat dated, unfortunately.

- Early Mesa development was done while Brian was part of the SSEC Visualization Project at the University of Wisconsin. He'd like to thank Bill Hibbard for letting him work on Mesa as part of that project.
- John Carmack of id Software, Inc. funded Keith Whitwell in 1999 in order to optimize Mesa's vertex transformation module. This is a very substantial piece of work.
- Precision Insight, Inc., VA Linux Systems, Inc., and most recently, Tungsten Graphics, Inc. have supported the ongoing development of Mesa.
- The Mesa website is hosted by sourceforge.net.
- The Mesa git repository is hosted by freedesktop.org.
- alt.software contributed the Direct3D driver.
- **Bernd Barsuhn** wrote the evaluator code for (splines, patches) in Mesa.
- **Bernhard Tschirren** wrote the Allegro DJGPP driver.
- **Bogdan Sikorski** wrote the GLU NURBS and polygon tessellator in Mesa.
- **Charlie Wallace** wrote the MS-DOS driver.
- **CJ Beyer** was the www.mesa3d.org webmaster.
- **Darren Abbott** provided the OS/2 driver.
- **David Bucciarelli** wrote and maintained the 3Dfx Glide driver. Thousands of Linux/Quake players thank David!
- **Gareth Hughes** wrote new GLU 1.2 Polygon Tessellation code (now superceded by SGI SI GLU).
- **Holger Waechtler** contributed AMD 3DNow! assembly code which accelerates vertex transformation in Mesa 3.1. Holger also implemented the GL_EXT_texture_env_combine extension.
- **Jeroen van der Zijp** and **Thorsten Ohl** contributed the Xt/Motif widget code.
- **John Stone** provided the multi-threading support in Mesa 3.0.
- **John Watson** assisted with web page design.
- **Josh Vanderhoof** contributed Intel x86 assembly code which accelerates vertex transformation in Mesa 3.x.
- **Jouk Jansen** contributed and continues to maintain the VMS support.
- **Karl Schultz** has been maintaining the Windows driver.
- **Keith Whitwell** has made extension contributions to Mesa since 1999.

- **Kendall Bennett** wrote the SciTech MGL driver.
- **Klaus Niederkrueger** contributed many improvements to Mesa's software rasterizer.
- **Mark Kilgard** contributed antialiased line improvements and several extensions.
- **Michael Pichler** contributed *many* bug fixes
- **Miklos Fazekas** wrote and maintains the Macintosh driver.
- **Pascal Thibaudeau** wrote the NeXT driver.
- **Pedro Vazquez** setup and maintains the Mesa Mailing list.
- **Randy Frank** contributed *many* bug fixes.
- **Stefan Zivkovic** wrote the Amiga driver.
- **Stephane Rehel** provided the Cygnus Win32 support
- **Ted Jump** maintained the makefiles and project files for Windows 95/98/NT compilation for some time.
- **Uwe Maurer** wrote the LibGGI driver for Mesa-3.0.
- **Victor Ng-Thow-Hing** wrote the Amiwin driver for the Amiga.

Apologies to anyone who's been omitted. Please send corrections and additions to Brian.

CHAPTER 8

Conformance

The SGI OpenGL conformance tests verify correct operation of OpenGL implementations. I, Brian Paul, have been given a copy of the tests for testing Mesa. The tests are not publicly available.

This file has the latest results of testing Mesa with the OpenGL 1.2 conformance tests. Testing with the preliminary OpenGL 1.3 tests has also been done. Mesa passes all the 1.3 tests.

The tests were run using the software X11 device driver on 24-bpp and 16-bpp displays.

Mesa 4.0 and later pass all conformance tests at all path levels. Note that this says nothing about the conformance of hardware drivers based upon Mesa.

```
COVERAGE TESTS
--------------

Test that all API functions accept the legal parameters and reject
illegal parameters.  The result of each test is either pass or fail.

% covgl
OpenGL Coverage Test.
Version 1.2

covgl passed.

covgl passed at 1.1 level.

covgl passed at 1.2 level.

covgl passed for ARB_multitexture.


% covglu
OpenGL GLU Coverage Test.
Version 1.3

covglu passed.

covglu passed at 1.1 level.


% covglx
OpenGL X Coverage Test.
Version 1.1.1

covglx passed.
```

```
% primtest -v
Open GL Primitives Test.
Version 1.2

[lots of output deleted]

292159 Combinations.
primtest passed.



GL CONFORMANCE TEST
===================

Render test images, read them back, then test for expected results.


------------------------------------------------------------------------
% conform -v 2

OpenGL Conformance Test
Version 1.2

Setup Report.
    Verbose level = 2.
    Random number seed = 1.
    Path inactive.

Visual Report.
    Display ID = 35. Indirect Rendering.
    Double Buffered.
    RGBA (5, 6, 5, 0).
    Stencil (8).
    Depth (16).
    Accumulation (16, 16, 16, 16).

Epsilon Report.
    zero error epsilon = 0.000122.
    RGBA error epsilon = 0.0324, 0.016, 0.0324, 0.000122.
    Depth buffer error epsilon = 0.000137.
    Stencil plane error epsilon = 0.00404.
    Accumulation error epsilon = 0.000137, 0.000137, 0.000137, 0.000137.

Default State test passed.
Must Pass test passed.
Divide By Zero test passed.
Viewport Clamp test passed.
Matrix Stack test passed.
Matrix Stack Mixing test passed.
Vertex Order test passed.
Transformations test passed.
Transformation Normal test passed.
Viewport Transformation test passed.
Buffer Clear test passed.
Buffer Corners test passed.
Buffer Color test passed.
```

```
Color Ramp test passed.
Mask test passed.
Buffer Invariance test passed.
Accumulation Buffer test passed.
Select test passed.
Feedback test passed.
Scissor test passed.
Alpha Plane Function test passed.
Stencil Plane Clear test passed.
Stencil Plane Corners test passed.
Stencil Plane Operation test passed.
Stencil Plane Function test passed.
Depth Buffer Clear test passed.
Depth Buffer Function test passed.
Blend test passed.
Dither test passed.
LogicOp Function test does not exist for an RGB visual.
DrawPixels test passed.
CopyPixels test passed.
Bitmap Rasterization test passed.
Point Rasterization test passed.
Anti-aliased Point test passed.
Line Rasterization test passed.
Line Stipple test passed.
Anti-aliased Line test passed.
Horizontal and Vertical Line test passed.
Triangle Rasterization test passed.
Triangle Tile test passed.
Triangle Stipple test passed.
Anti-aliased Triangles test passed.
Quad Rasterization test passed.
Polygon Face test passed.
Polygon Cull test passed.
Polygon Stipple test passed.
Polygon Edge test passed.
Ambient Material test passed.
Ambient Scene test passed.
Attenuation Position test passed.
Diffuse Light test passed.
Diffuse Material test passed.
Diffuse Material Normal test passed.
Diffuse Material Positioning test passed.
Emissive Material test passed.
Specular Exponent test passed.
Specular Exponent Normal test passed.
Specular Local Eye Half Angle test passed.
Specular Light test passed.
Specular Material test passed.
Specular Normal test passed.
Spot Positioning test passed.
Spot Exponent and Positioning test passed.
Spot Exponent and Direction test passed.
Fog Exponential test passed.
Fog Linear test passed.
Texture Decal test passed.
Texture Border test passed.
Mipmaps Selection test passed.
Mipmaps Interpolation test passed.
```

```
Display Lists test passed.
Evaluator test passed.
Evaluator Color test passed.
Texture Edge Clamp test passed.
Packed Pixels test passed.
Texture LOD test passed.
Rescale Normal test passed.
Color Table test passed.
Convolution test passed.
Convolution Border test passed.
Histogram test passed.
MinMax test passed.
MultiTexture test passed.

Conform passed.

-----------------------------------------------------------------
% conform -v 2 -p 1

OpenGL Conformance Test
Version 1.2

Setup Report.
    Verbose level = 2.
    Random number seed = 1.
    Path level = 1.

Visual Report.
    Display ID = 35. Indirect Rendering.
    Double Buffered.
    RGBA (5, 6, 5, 0).
    Stencil (8).
    Depth (16).
    Accumulation (16, 16, 16, 16).

Epsilon Report.
    zero error epsilon = 0.000122.
    RGBA error epsilon = 0.0324, 0.016, 0.0324, 0.000122.
    Depth buffer error epsilon = 0.000137.
    Stencil plane error epsilon = 0.00404.
    Accumulation error epsilon = 0.000137, 0.000137, 0.000137, 0.000137.

Default State test passed.
Must Pass test passed.
Divide By Zero test passed.
Viewport Clamp test passed.
Matrix Stack test passed.
Matrix Stack Mixing test passed.
Vertex Order test passed.
Transformations test passed.
Transformation Normal test passed.
Viewport Transformation test passed.
Buffer Clear test passed.
Buffer Corners test passed.
Buffer Color test passed.
Color Ramp test passed.
Mask test passed.
Buffer Invariance test passed.
```

```
Accumulation Buffer test passed.
Select test passed.
Feedback test passed.
Scissor test passed.
Alpha Plane Function test passed.
Stencil Plane Clear test passed.
Stencil Plane Corners test passed.
Stencil Plane Operation test passed.
Stencil Plane Function test passed.
Depth Buffer Clear test passed.
Depth Buffer Function test passed.
Blend test passed.
Dither test passed.
LogicOp Function test does not exist for an RGB visual.
DrawPixels test passed.
CopyPixels test passed.
Bitmap Rasterization test passed.
Point Rasterization test passed.
Anti-aliased Point test passed.
Line Rasterization test passed.
Line Stipple test passed.
Anti-aliased Line test passed.
Horizontal and Vertical Line test passed.
Triangle Rasterization test passed.
Triangle Tile test passed.
Triangle Stipple test passed.
Anti-aliased Triangles test passed.
Quad Rasterization test passed.
Polygon Face test passed.
Polygon Cull test passed.
Polygon Stipple test passed.
Polygon Edge test passed.
Ambient Material test passed.
Ambient Scene test passed.
Attenuation Position test passed.
Diffuse Light test passed.
Diffuse Material test passed.
Diffuse Material Normal test passed.
Diffuse Material Positioning test passed.
Emissive Material test passed.
Specular Exponent test passed.
Specular Exponent Normal test passed.
Specular Local Eye Half Angle test passed.
Specular Light test passed.
Specular Material test passed.
Specular Normal test passed.
Spot Positioning test passed.
Spot Exponent and Positioning test passed.
Spot Exponent and Direction test passed.
Fog Exponential test passed.
Fog Linear test passed.
Texture Decal test passed.
Texture Border test passed.
Mipmaps Selection test passed.
Mipmaps Interpolation test passed.
Display Lists test passed.
Evaluator test passed.
Evaluator Color test passed.
```

```
Texture Edge Clamp test passed.
Packed Pixels test passed.
Texture LOD test passed.
Rescale Normal test passed.
Color Table test passed.
Convolution test passed.
Convolution Border test passed.
Histogram test passed.
MinMax test passed.
MultiTexture test passed.

Conform passed.

-------------------------------------------------------------------------
% conform -v 2 -p 2

OpenGL Conformance Test
Version 1.2

Setup Report.
    Verbose level = 2.
    Random number seed = 1.
    Path level = 2.

Visual Report.
    Display ID = 35. Indirect Rendering.
    Double Buffered.
    RGBA (5, 6, 5, 0).
    Stencil (8).
    Depth (16).
    Accumulation (16, 16, 16, 16).

Epsilon Report.
    zero error epsilon = 0.000122.
    RGBA error epsilon = 0.0324, 0.016, 0.0324, 0.000122.
    Depth buffer error epsilon = 0.000137.
    Stencil plane error epsilon = 0.00404.
    Accumulation error epsilon = 0.000137, 0.000137, 0.000137, 0.000137.

Default State test passed.
Must Pass test passed.
Divide By Zero test passed.
Viewport Clamp test passed.
Matrix Stack test passed.
Matrix Stack Mixing test passed.
Vertex Order test passed.
Transformations test passed.
Transformation Normal test passed.
Viewport Transformation test passed.
Buffer Clear test passed.
Buffer Corners test passed.
Buffer Color test passed.
Color Ramp test passed.
Mask test passed.
Buffer Invariance test passed.
Accumulation Buffer test passed.
Select test passed.
Feedback test passed.
```

```
Scissor test passed.
Alpha Plane Function test passed.
Stencil Plane Clear test passed.
Stencil Plane Corners test passed.
Stencil Plane Operation test passed.
Stencil Plane Function test passed.
Depth Buffer Clear test passed.
Depth Buffer Function test passed.
Blend test passed.
Dither test passed.
LogicOp Function test does not exist for an RGB visual.
DrawPixels test passed.
CopyPixels test passed.
Bitmap Rasterization test passed.
Point Rasterization test passed.
Anti-aliased Point test passed.
Line Rasterization test passed.
Line Stipple test passed.
Anti-aliased Line test passed.
Horizontal and Vertical Line test passed.
Triangle Rasterization test passed.
Triangle Tile test passed.
Triangle Stipple test passed.
Anti-aliased Triangles test passed.
Quad Rasterization test passed.
Polygon Face test passed.
Polygon Cull test passed.
Polygon Stipple test passed.
Polygon Edge test passed.
Ambient Material test passed.
Ambient Scene test passed.
Attenuation Position test passed.
Diffuse Light test passed.
Diffuse Material test passed.
Diffuse Material Normal test passed.
Diffuse Material Positioning test passed.
Emissive Material test passed.
Specular Exponent test passed.
Specular Exponent Normal test passed.
Specular Local Eye Half Angle test passed.
Specular Light test passed.
Specular Material test passed.
Specular Normal test passed.
Spot Positioning test passed.
Spot Exponent and Positioning test passed.
Spot Exponent and Direction test passed.
Fog Exponential test passed.
Fog Linear test passed.
Texture Decal test passed.
Texture Border test passed.
Mipmaps Selection test passed.
Mipmaps Interpolation test passed.
Display Lists test passed.
Evaluator test passed.
Evaluator Color test passed.
Texture Edge Clamp test passed.
Packed Pixels test passed.
Texture LOD test passed.
```

```
Rescale Normal test passed.
Color Table test passed.
Convolution test passed.
Convolution Border test passed.
Histogram test passed.
MinMax test passed.
MultiTexture test passed.

Conform passed.

-------------------------------------------------------------------------
% conform -v 2 -p 3

OpenGL Conformance Test
Version 1.2

Setup Report.
    Verbose level = 2.
    Random number seed = 1.
    Path level = 3.

Visual Report.
    Display ID = 35. Indirect Rendering.
    Double Buffered.
    RGBA (5, 6, 5, 0).
    Stencil (8).
    Depth (16).
    Accumulation (16, 16, 16, 16).

Epsilon Report.
    zero error epsilon = 0.000122.
    RGBA error epsilon = 0.0324, 0.016, 0.0324, 0.000122.
    Depth buffer error epsilon = 0.000137.
    Stencil plane error epsilon = 0.00404.
    Accumulation error epsilon = 0.000137, 0.000137, 0.000137, 0.000137.

Default State test passed.
Must Pass test passed.
Divide By Zero test passed.
Viewport Clamp test passed.
Matrix Stack test passed.
Matrix Stack Mixing test passed.
Vertex Order test passed.
Transformations test passed.
Transformation Normal test passed.
Viewport Transformation test passed.
Buffer Clear test passed.
Buffer Corners test passed.
Buffer Color test passed.
Color Ramp test passed.
Mask test passed.
Buffer Invariance test passed.
Accumulation Buffer test passed.
Select test passed.
Feedback test passed.
Scissor test passed.
Alpha Plane Function test passed.
Stencil Plane Clear test passed.
```

```
Stencil Plane Corners test passed.
Stencil Plane Operation test passed.
Stencil Plane Function test passed.
Depth Buffer Clear test passed.
Depth Buffer Function test passed.
Blend test passed.
Dither test passed.
LogicOp Function test does not exist for an RGB visual.
DrawPixels test passed.
CopyPixels test passed.
Bitmap Rasterization test passed.
Point Rasterization test passed.
Anti-aliased Point test passed.
Line Rasterization test passed.
Line Stipple test passed.
Anti-aliased Line test passed.
Horizontal and Vertical Line test passed.
Triangle Rasterization test passed.
Triangle Tile test passed.
Triangle Stipple test passed.
Anti-aliased Triangles test passed.
Quad Rasterization test passed.
Polygon Face test passed.
Polygon Cull test passed.
Polygon Stipple test passed.
Polygon Edge test passed.
Ambient Material test passed.
Ambient Scene test passed.
Attenuation Position test passed.
Diffuse Light test passed.
Diffuse Material test passed.
Diffuse Material Normal test passed.
Diffuse Material Positioning test passed.
Emissive Material test passed.
Specular Exponent test passed.
Specular Exponent Normal test passed.
Specular Local Eye Half Angle test passed.
Specular Light test passed.
Specular Material test passed.
Specular Normal test passed.
Spot Positioning test passed.
Spot Exponent and Positioning test passed.
Spot Exponent and Direction test passed.
Fog Exponential test passed.
Fog Linear test passed.
Texture Decal test passed.
Texture Border test passed.
Mipmaps Selection test passed.
Mipmaps Interpolation test passed.
Display Lists test passed.
Evaluator test passed.
Evaluator Color test passed.
Texture Edge Clamp test passed.
Packed Pixels test passed.
Texture LOD test passed.
Rescale Normal test passed.
Color Table test passed.
Convolution test passed.
```

```
Convolution Border test passed.
Histogram test passed.
MinMax test passed.
MultiTexture test passed.

Conform passed.

-------------------------------------------------------------------------
% conform -v 2 -p 4

OpenGL Conformance Test
Version 1.2

Setup Report.
    Verbose level = 2.
    Random number seed = 1.
    Path level = 4.

Visual Report.
    Display ID = 35. Indirect Rendering.
    Double Buffered.
    RGBA (5, 6, 5, 0).
    Stencil (8).
    Depth (16).
    Accumulation (16, 16, 16, 16).

Epsilon Report.
    zero error epsilon = 0.000122.
    RGBA error epsilon = 0.0324, 0.016, 0.0324, 0.000122.
    Depth buffer error epsilon = 0.000137.
    Stencil plane error epsilon = 0.00404.
    Accumulation error epsilon = 0.000137, 0.000137, 0.000137, 0.000137.

Default State test passed.
Must Pass test passed.
Divide By Zero test passed.
Viewport Clamp test passed.
Matrix Stack test passed.
Matrix Stack Mixing test passed.
Vertex Order test passed.
Transformations test passed.
Transformation Normal test passed.
Viewport Transformation test passed.
Buffer Clear test passed.
Buffer Corners test passed.
Buffer Color test passed.
Color Ramp test passed.
Mask test passed.
Buffer Invariance test passed.
Accumulation Buffer test passed.
Select test passed.
Feedback test passed.
Scissor test passed.
Alpha Plane Function test passed.
Stencil Plane Clear test passed.
Stencil Plane Corners test passed.
Stencil Plane Operation test passed.
Stencil Plane Function test passed.
```

```
Depth Buffer Clear test passed.
Depth Buffer Function test passed.
Blend test passed.
Dither test passed.
LogicOp Function test does not exist for an RGB visual.
DrawPixels test passed.
CopyPixels test passed.
Bitmap Rasterization test passed.
Point Rasterization test passed.
Anti-aliased Point test passed.
Line Rasterization test passed.
Line Stipple test passed.
Anti-aliased Line test passed.
Horizontal and Vertical Line test passed.
Triangle Rasterization test passed.
Triangle Tile test passed.
Triangle Stipple test passed.
Anti-aliased Triangles test passed.
Quad Rasterization test passed.
Polygon Face test passed.
Polygon Cull test passed.
Polygon Stipple test passed.
Polygon Edge test passed.
Ambient Material test passed.
Ambient Scene test passed.
Attenuation Position test passed.
Diffuse Light test passed.
Diffuse Material test passed.
Diffuse Material Normal test passed.
Diffuse Material Positioning test passed.
Emissive Material test passed.
Specular Exponent test passed.
Specular Exponent Normal test passed.
Specular Local Eye Half Angle test passed.
Specular Light test passed.
Specular Material test passed.
Specular Normal test passed.
Spot Positioning test passed.
Spot Exponent and Positioning test passed.
Spot Exponent and Direction test passed.
Fog Exponential test passed.
Fog Linear test passed.
Texture Decal test passed.
Texture Border test passed.
Mipmaps Selection test passed.
Mipmaps Interpolation test passed.
Display Lists test passed.
Evaluator test passed.
Evaluator Color test passed.
Texture Edge Clamp test passed.
Packed Pixels test passed.
Texture LOD test passed.
Rescale Normal test passed.
Color Table test passed.
Convolution test passed.
Convolution Border test passed.
Histogram test passed.
MinMax test passed.
```

```
MultiTexture test passed.

Conform passed.



GLX CONFORMANCE TEST
====================

% conformx -v 2

OpenGL X Conformance Test
Version 1.1.1

Setup Report.
    Verbose level = 2.
    Random number seed = 1.
    Path inactive.

Visual Report.
    Display ID = 34. Direct Rendering.
    Double Buffered.
    RGBA (8, 8, 8, 0).
    Stencil (8).
    Depth (16).
    Accumulation (16, 16, 16, 16).

Epsilon Report.
    zero error epsilon = 0.000122.
    RGBA error epsilon = 0.00404, 0.00404, 0.00404, 0.000122.
    Depth buffer error epsilon = 0.000137.
    Stencil plane error epsilon = 0.00404.
    Accumulation error epsilon = 0.000137, 0.000137, 0.000137, 0.000137.

Default State test passed.
glReadPixels() test passed.
Font test passed.

Conformx passed.
```

NOTE: conformx passes for all machine path levels (-p option).

CHAPTER 9

Downloading

Primary Mesa download site: ftp.freedesktop.org (FTP) or mesa.freedesktop.org (HTTP).

When a new release is coming, release candidates (betas) may be found here.

Unpacking

Mesa releases are available in three formats: .tar.bz2, .tar.gz, and .zip

To unpack .tar.gz files:

```
tar zxf MesaLib-x.y.z.tar.gz
```

or:

```
gzcat MesaLib-x.y.z.tar.gz | tar xf -
```

or:

```
gunzip MesaLib-x.y.z.tar.gz ; tar xf MesaLib-x.y.z.tar
```

To unpack .tar.bz2 files:

```
bunzip2 -c MesaLib-x.y.z.tar.gz | tar xf -
```

To unpack .zip files:

```
unzip MesaLib-x.y.z.zip
```

Contents

After unpacking you'll have these files and directories (among others):

```
Makefile    - top-level Makefile for most systems
configs/    - makefile parameter files for various systems
include/    - GL header (include) files
bin/        - shell scripts for making shared libraries, etc
docs/       - documentation
src/        - source code for libraries
src/mesa    - sources for the main Mesa library and device drivers
src/gallium     - sources for Gallium and Gallium drivers
src/glx     - sources for building libGL with full GLX and DRI support
```

Proceed to the compilation and installation instructions.

Demos, GLUT, and GLU

A package of SGI's GLU library is available here

A package of Mark Kilgard's GLUT library is available here

The Mesa demos collection is available here

In the past, GLUT, GLU and the Mesa demos were released in conjunction with Mesa releases. But since GLUT, GLU and the demos change infrequently, they were split off into their own git repositories: GLUT, GLU and Demos,

CHAPTER 10

Compiling and Installing

1. *Prerequisites for building*
    - *General prerequisites*
    - *For DRI and hardware acceleration*
2. *Building with autoconf (Linux/Unix/X11)*
3. *Building with SCons (Windows/Linux)*
4. *Building for other systems*
5. *Library Information*
6. *Building OpenGL programs with pkg-config*

1. Prerequisites for building

1.1 General

- Python - Python is required. Version 2.6.4 or later should work.
- Python Mako module - Python Mako module is required. Version 0.3.4 or later should work.
- SCons is required for building on Windows and optional for Linux (it's an alternative to autoconf/automake.)
- lex / yacc - for building the GLSL compiler.
  On Linux systems, flex and bison are used. Versions 2.5.35 and 2.4.1, respectively, (or later) should work.
  On Windows with MinGW, install flex and bison with:

```
mingw-get install msys-flex msys-bison
```

  For MSVC on Windows, install Win flex-bison.
- For building on Windows, Microsoft Visual Studio 2013 or later is required.

1.2 For DRI and hardware acceleration

The following are required for DRI-based hardware acceleration with Mesa:

- dri2proto version 2.6 or later
- libDRM latest version
- Xorg server version 1.5 or later

- Linux 2.6.28 or later

If you're using a fedora distro the following command should install all the needed dependencies:

```
sudo yum install flex bison imake libtool xorg-x11-proto-devel libdrm-devel \
gcc-c++ xorg-x11-server-devel libXi-devel libXmu-devel libXdamage-devel git \
expat-devel llvm-devel python-mako
```

2. Building with autoconf (Linux/Unix/X11)

The primary method to build Mesa on Unix systems is with autoconf.

The general approach is the standard:

```
./configure
make
sudo make install
```

But please read the detailed autoconf instructions for more details.

3. Building with SCons (Windows/Linux)

To build Mesa with SCons on Linux or Windows do

```
scons
```

The build output will be placed in build/*platform-machine-debug*/..., where *platform* is for example linux or windows, *machine* is x86 or x86_64, optionally followed by -debug for debug builds.

To build Mesa with SCons for Windows on Linux using the MinGW crosscompiler toolchain do

```
scons platform=windows toolchain=crossmingw machine=x86 libgl-gdi
```

This will create:

- build/windows-x86-debug/gallium/targets/libgl-gdi/opengl32.dll — Mesa + Gallium + softpipe (or llvmpipe), binary compatible with Windows's opengl32.dll

Put them all in the same directory to test them.

4. Building for other systems

Documentation for other environments (some may be very out of date):

- README.VMS - VMS
- README.CYGWIN - Cygwin
- README.WIN32 - Win32

5. Library Information

When compilation has finished, look in the top-level `lib/` (or `lib64/`) directory. You'll see a set of library files similar to this:

```
lrwxrwxrwx    1 brian    users          10 Mar 26 07:53 libGL.so -> libGL.so.1*
lrwxrwxrwx    1 brian    users          19 Mar 26 07:53 libGL.so.1 -> libGL.so.1.5.060100*
-rwxr-xr-x    1 brian    users     3375861 Mar 26 07:53 libGL.so.1.5.060100*
lrwxrwxrwx    1 brian    users          14 Mar 26 07:53 libOSMesa.so -> libOSMesa.so.6*
lrwxrwxrwx    1 brian    users          23 Mar 26 07:53 libOSMesa.so.6 -> libOSMesa.so.6.1.060100*
-rwxr-xr-x    1 brian    users       23871 Mar 26 07:53 libOSMesa.so.6.1.060100*
```

libGL is the main OpenGL library (i.e. Mesa).
libOSMesa is the OSMesa (Off-Screen) interface library.

If you built the DRI hardware drivers, you'll also see the DRI drivers:

```
-rwxr-xr-x   1 brian users 16895413 Jul 21 12:11 i915_dri.so
-rwxr-xr-x   1 brian users 16895413 Jul 21 12:11 i965_dri.so
-rwxr-xr-x   1 brian users 11849858 Jul 21 12:12 r200_dri.so
-rwxr-xr-x   1 brian users 11757388 Jul 21 12:12 radeon_dri.so
```

If you built with Gallium support, look in lib/gallium/ for Gallium-based versions of libGL and device drivers.

6. Building OpenGL programs with pkg-config

Running `make install` will install package configuration files for the pkg-config utility.

When compiling your OpenGL application you can use pkg-config to determine the proper compiler and linker flags.

For example, compiling and linking a GLUT application can be done with:

```
gcc `pkg-config --cflags --libs glut` mydemo.c -o mydemo
```

CHAPTER 11

Compilation and Installation using Autoconf

1. Basic Usage

The autoconf generated configure script can be used to guess your platform and change various options for building Mesa. To use the configure script, type:

```
./configure
```

To see a short description of all the options, type `./configure --help`. If you are using a development snapshot and the configure script does not exist, type `./autogen.sh` to generate it first. If you know the options you want to pass to `configure`, you can pass them to `autogen.sh`. It will run `configure` with these options after it is generated. Once you have run `configure` and set the options to your preference, type:

```
make
```

This will produce libGL.so and several other libraries depending on the options you have chosen. Later, if you want to rebuild for a different configuration run `make realclean` before rebuilding.

Some of the generic autoconf options are used with Mesa:

`--prefix=PREFIX` This is the root directory where files will be installed by `make install`. The default is `/usr/local`.

`--exec-prefix=EPREFIX` This is the root directory where architecture-dependent files will be installed. In Mesa, this is only used to derive the directory for the libraries. The default is `${prefix}`.

`--libdir=LIBDIR` This option specifies the directory where the GL libraries will be installed. The default is `${exec_prefix}/lib`. It also serves as the name of the library staging area in the source tree. For instance, if the option `--libdir=/usr/local/lib64` is used, the libraries will be created in a `lib64` directory at the top of the Mesa source tree.

`--sysconfdir=DIR` This option specifies the directory where the configuration files will be installed. The default is `${prefix}/etc`. Currently there's only one config file provided when dri drivers are enabled - it's `drirc`.

--enable-static, --disable-shared By default, Mesa will build shared libraries. Either of these options will force static libraries to be built. It is not currently possible to build static and shared libraries in a single pass.

CC, CFLAGS, CXX, CXXFLAGS These environment variables control the C and C++ compilers used during the build. By default, `gcc` and `g++` are used and the debug/optimisation level is left unchanged.

LDFLAGS An environment variable specifying flags to pass when linking programs. These should be empty and `PKG_CONFIG_PATH` is recommended to be used instead. If needed it can be used to direct the linker to use libraries in nonstandard directories. For example, `LDFLAGS="-L/usr/X11R6/lib"`.

PKG_CONFIG_PATH The `pkg-config` utility is a hard requirement for cofiguring and building mesa. It is used to search for external libraries on the system. This environment variable is used to control the search path for `pkg-config`. For instance, setting `PKG_CONFIG_PATH=/usr/X11R6/lib/pkgconfig` will search for package metadata in `/usr/X11R6` before the standard directories.

There are also a few general options for altering the Mesa build:

--enable-debug This option will enable compiler options and macros to aid in debugging the Mesa libraries.

--disable-asm There are assembly routines available for a few architectures. These will be used by default if one of these architectures is detected. This option ensures that assembly will not be used.

`--build= --host=`

By default, the build will compile code for the architecture that it's running on. In order to build cross-compile Mesa on a x86-64 machine that is to run on a i686, one would need to set the options to:

```
--build=x86_64-pc-linux-gnu --host=i686-pc-linux-gnu
```

Note that these can vary from distribution to distribution. For more information check with the autoconf manual. Note that you will need to correctly set `PKG_CONFIG_PATH` as well.

In some cases a single compiler is capable of handling both architectures (multilib) in that case one would need to set the `CC,CXX` variables appending the correct machine options. Seek your compiler documentation for further information - gcc machine dependent options

In addition to specifying correct `PKG_CONFIG_PATH` for the target architecture, the following should be sufficient to configure multilib Mesa

```
./configure CC="gcc -m32" CXX="g++ -m32" --build=x86_64-pc-linux-gnu
--host=i686-pc-linux-gnu ...
```

2. Driver Options

There are several different driver modes that Mesa can use. These are described in more detail in the basic installation instructions. The Mesa driver is controlled through the configure options `--enable-xlib-glx`, `--enable-osmesa`, and `--enable-dri`.

Xlib

It uses Xlib as a software renderer to do all rendering. It corresponds to the option `--enable-xlib-glx`. The libX11 and libXext libraries, as well as the X11 development headers, will be need to support the Xlib driver.

DRI

This mode uses the DRI hardware drivers for accelerated OpenGL rendering. Enable the DRI drivers with the option `--enable-dri`. See the basic installation instructions for details on prerequisites for the DRI drivers.

`--with-dri-driverdir=DIR` This option specifies the location the DRI drivers will be installed to and the location libGL will search for DRI drivers. The default is `${libdir}/dri`. `--with-dri-drivers=DRIVER,DRIVER,...` This option allows a specific set of DRI drivers to be built. For example, `--with-dri-drivers="swrast,i965,radeon,nouveau"`. By default, the drivers will be chosen depending on the target platform. See the directory `src/mesa/drivers/dri` in the source tree for available drivers. Beware that the swrast DRI driver is used by both libGL and the X.Org xserver GLX module to do software rendering, so you may run into problems if it is not available. `--disable-driglx-direct` Disable direct rendering in GLX. Normally, direct hardware rendering through the DRI drivers and indirect software rendering are enabled in GLX. This option disables direct rendering entirely. It can be useful on architectures where kernel DRM modules are not available. `--enable-glx-tls` Enable Thread Local Storage (TLS) in GLX. `--with-expat=DIR` **DEPRECATED**, use `PKG_CONFIG_PATH` instead.

The DRI-enabled libGL uses expat to parse the DRI configuration files in `${sysconfdir}/drirc` and `~/.drirc`. This option allows a specific expat installation to be used. For example, `--with-expat=/usr/local` will search for expat headers and libraries in `/usr/local/include` and `/usr/local/lib`, respectively. .. rubric:: OSMesa

name osmesa

No libGL is built in this mode. Instead, the driver code is built into the Off-Screen Mesa (OSMesa) library. See the Off-Screen Rendering page for more details. It corresponds to the option `--enable-osmesa`.

`--with-osmesa-bits=BITS` This option allows the size of the color channel in bits to be specified. By default, an 8-bit channel will be used, and the driver will be named libOSMesa. Other options are 16- and 32-bit color channels, which will add the bit size to the library name. For example, `--with-osmesa-bits=16` will create the libOSMesa16 library with a 16-bit color channel. .. rubric:: 3. Library Options

name library

The configure script provides more fine grained control over the GL libraries that will be built. More details on the specific GL libraries can be found in the basic installation instructions.

CHAPTER 12

Precompiled Libraries

In general, precompiled Mesa libraries are not available.

However, some Linux distros (such as Ubuntu) seem to closely track Mesa and often have the latest Mesa release available as an update.

CHAPTER 13

Mailing Lists

There are four Mesa 3D / DRI mailing lists:

- mesa-users - intended for end-users of Mesa and DRI drivers. Newbie questions are OK, but please try the general OpenGL resources and Mesa/DRI documentation first.
- mesa-dev - for Mesa, Gallium and DRI development discussion. Not for beginners.
- mesa-commit - relays git check-in messages (for developers). In general, people should not post to this list.
- mesa-announce - announcements of new Mesa versions are sent to this list. Very low traffic.
- piglit - for Piglit (OpenGL driver testing framework) discussion.

NOTE: You **must** subscribe to these lists in order to post to them. If you try to post to a list and you're not a subscriber (or if you try to post from a different email address than you subscribed with) your posting will be held for an indefinite period or may be discarded entirely.

Follow the links above for list archives.

The old Mesa lists hosted at SourceForge are no longer in use. The archives are still available, however: mesa3d-announce, mesa3d-users, mesa3d-dev.

For mailing lists about Direct Rendering Modules (drm) in Linux/BSD kernels, see the DRI wiki.

IRC

join #dri-devel channel on irc.freenode.net

OpenGL Forums

Here are some other OpenGL-related forums you might find useful:

- OpenGL discussion forums at www.opengl.org
- Usenet newsgroups:
  - comp.graphics.algorithms
  - comp.graphics.api.opengl
  - comp.os.linux.x

CHAPTER 14

Bug Database

The Mesa bug database is hosted on freedesktop.org. The old bug database on SourceForge is no longer used.

To file a Mesa bug, go to Bugzilla on freedesktop.org

Please follow these bug reporting guidelines:

- Check if a new version of Mesa is available which might have fixed the problem.
- Check if your bug is already reported in the database.
- Monitor your bug report for requests for additional information, etc.
- If you're reporting a crash, try to use your debugger (gdb) to get a stack trace. Also, recompile Mesa in debug mode to get more detailed information.
- Describe in detail how to reproduce the bug, especially with games and applications that the Mesa developers might not be familiar with.
- Provide a simple GLUT-based test program if possible

Bug reports will automatically be forwarded by bugzilla to the Mesa developer's mailing list.

The easier a bug is to reproduce, the sooner it will be fixed. Please do everything you can to facilitate quickly fixing bugs. If your bug report is vague or your test program doesn't compile easily, the problem may not be fixed very quickly.

CHAPTER 15

Webmaster

If you have problems, edits or additions for this website send them to Brian (*brian.e.paul gmail.com*).

Mark Manning made the frame-based layout for the website. Brian's modified it a lot since then.

CHAPTER 16

Shading Language Support

This page describes the features and status of Mesa's support for the OpenGL Shading Language.

Contents

Environment Variables

The **MESA_GLSL** environment variable can be set to a comma-separated list of keywords to control some aspects of the GLSL compiler and shader execution. These are generally used for debugging.

- **dump** - print GLSL shader code to stdout at link time
- **log** - log all GLSL shaders to files. The filenames will be "shader_X.vert" or "shader_X.frag" where X the shader ID.
- **nopt** - disable compiler optimizations
- **opt** - force compiler optimizations
- **uniform** - print message to stdout when glUniform is called
- **nopvert** - force vertex shaders to be a simple shader that just transforms the vertex position with ftransform() and passes through the color and texcoord[0] attributes.
- **nopfrag** - force fragment shader to be a simple shader that passes through the color attribute.
- **useprog** - log glUseProgram calls to stderr

Example: export MESA_GLSL=dump,nopt

Shaders can be dumped and replaced on runtime for debugging purposes. Mesa needs to be configured with '–with-sha1' to enable this functionality. This feature is not currently supported by SCons build. This is controlled via following environment variables:

- **MESA_SHADER_DUMP_PATH** - path where shader sources are dumped
- **MESA_SHADER_READ_PATH** - path where replacement shaders are read

Note, path set must exist before running for dumping or replacing to work. When both are set, these paths should be different so the dumped shaders do not clobber the replacement shaders. .. rubric:: GLSL Version

name support

The GLSL compiler currently supports version 3.30 of the shading language.

Several GLSL extensions are also supported:

- GL_ARB_draw_buffers
- GL_ARB_fragment_coord_conventions
- GL_ARB_shader_bit_encoding

Unsupported Features

XXX update this section

The following features of the shading language are not yet fully supported in Mesa:

- Linking of multiple shaders does not always work. Currently, linking is implemented through shader concatenation and re-compiling. This doesn't always work because of some #pragma and preprocessor issues.
- The gl_Color and gl_SecondaryColor varying vars are interpolated without perspective correction

All other major features of the shading language should function.

Implementation Notes

- Shading language programs are compiled into low-level programs very similar to those of GL_ARB_vertex/fragment_program.
- All vector types (vec2, vec3, vec4, bvec2, etc) currently occupy full float[4] registers.
- Float constants and variables are packed so that up to four floats can occupy one program parameter/register.
- All function calls are inlined.
- Shaders which use too many registers will not compile.
- The quality of generated code is pretty good, register usage is fair.
- Shader error detection and reporting of errors (InfoLog) is not very good yet.
- The ftransform() function doesn't necessarily match the results of fixed-function transformation.

These issues will be addressed/resolved in the future.

Programming Hints

- Use the built-in library functions whenever possible. For example, instead of writing this:

```
float x = 1.0 / sqrt(y);
```

Write this:

```
    float x = inversesqrt(y);
```

Stand-alone GLSL Compiler

The stand-alone GLSL compiler program can be used to compile GLSL shaders into low-level GPU code.

This tool is useful for:

- Inspecting GPU code to gain insight into compilation
- Generating initial GPU code for subsequent hand-tuning
- Debugging the GLSL compiler itself

After building Mesa, the compiler can be found at src/glsl/glsl_compiler

Here's an example of using the compiler to compile a vertex shader and emit GL_ARB_vertex_program-style instructions:

```
src/glsl/glsl_compiler --dump-ast myshader.vert
```

Options include

- **–dump-ast** - dump GPU code
- **–dump-hir** - dump high-level IR code
- **–dump-lir** - dump low-level IR code
- **–link** - ???

Compiler Implementation

The source code for Mesa's shading language compiler is in the `src/glsl/` directory.

XXX provide some info about the compiler....

The final vertex and fragment programs may be interpreted in software (see prog_execute.c) or translated into a specific hardware architecture (see drivers/dri/i915/i915_fragprog.c for example).

Compiler Validation

Developers working on the GLSL compiler should test frequently to avoid regressions.

The Piglit project has many GLSL tests.

The Mesa demos repository also has some good GLSL tests.

CHAPTER 17

Mesa EGL

The current version of EGL in Mesa implements EGL 1.4. More information about EGL can be found at http://www.khronos.org/egl/.

The Mesa's implementation of EGL uses a driver architecture. The main library (`libEGL`) is window system neutral. It provides the EGL API entry points and helper functions for use by the drivers. Drivers are dynamically loaded by the main library and most of the EGL API calls are directly dispatched to the drivers.

The driver in use decides the window system to support.

Build EGL

1. Run `configure` with the desired client APIs and enable the driver for your hardware. For example

```
$ ./configure --enable-gles1 --enable-gles2 \
              --with-dri-drivers=... \
              --with-gallium-drivers=...
```

   The main library and OpenGL is enabled by default. The first two options above enables OpenGL ES 1.x and 2.x. The last two options enables the listed classic and and Gallium drivers respectively.

2. Build and install Mesa as usual.

In the given example, it will build and install `libEGL`, `libGL`, `libGLESv1_CM`, `libGLESv2`, and one or more EGL drivers.

Configure Options

There are several options that control the build of EGL at configuration time

`--enable-egl` By default, EGL is enabled. When disabled, the main library and the drivers will not be built.

`--with-egl-driver-dir` The directory EGL drivers should be installed to. If not specified, EGL drivers will be installed to `${libdir}/egl`.

`--with-egl-platforms` List the platforms (window systems) to support. Its argument is a comma separated string such as `--with-egl-platforms=x11,drm`. It decides the platforms a driver may support. The first listed platform is also used by the main library to decide the native platform: the platform the EGL native types such as `EGLNativeDisplayType` or `EGLNativeWindowType` defined for.

The available platforms are `x11`, `drm`, `wayland`, `surfaceless`, `android`, and `haiku`. The `android` platform can either be built as a system component, part of AOSP, using `Android.mk` files, or cross-compiled

using appropriate `configure` options. The `haiku` platform can only be built with SCons. Unless for special needs, the build system should select the right platforms automatically.

`--enable-gles1` `--enable-gles2`

These options enable OpenGL ES support in OpenGL. The result is one big internal library that supports multiple APIs.

`--enable-shared-glapi` By default, `libGL` has its own copy of `libglapi`. This options makes `libGL` use the shared `libglapi`. This is required if applications mix OpenGL and OpenGL ES.

Use EGL

Demos

There are demos for the client APIs supported by EGL. They can be found in mesa/demos repository.

Environment Variables

There are several environment variables that control the behavior of EGL at runtime

`EGL_DRIVERS_PATH` By default, the main library will look for drivers in the directory where the drivers are installed to. This variable specifies a list of colon-separated directories where the main library will look for drivers, in addition to the default directory. This variable is ignored for setuid/setgid binaries.

This variable is usually set to test an uninstalled build. For example, one may set

```
$ export LD_LIBRARY_PATH=$mesa/lib
$ export EGL_DRIVERS_PATH=$mesa/lib/egl
```

to test a build without installation

`EGL_DRIVER` This variable specifies a full path to or the name of an EGL driver. It forces the specified EGL driver to be loaded. It comes in handy when one wants to test a specific driver. This variable is ignored for setuid/setgid binaries.

`EGL_PLATFORM` This variable specifies the native platform. The valid values are the same as those for `--with-egl-platforms`. When the variable is not set, the main library uses the first platform listed in `--with-egl-platforms` as the native platform.

Extensions like `EGL_MESA_drm_display` define new functions to create displays for non-native platforms. These extensions are usually used by applications that support non-native platforms. Setting this variable is probably required only for some of the demos found in mesa/demo repository.

`EGL_LOG_LEVEL` This changes the log level of the main library and the drivers. The valid values are: `debug`, `info`, `warning`, and `fatal`.

EGL Drivers

`egl_dri2` This driver supports both `x11` and `drm` platforms. It functions as a DRI driver loader. For `x11` support, it talks to the X server directly using (XCB-)DRI2 protocol.

This driver can share DRI drivers with `libGL`.

Packaging

The ABI between the main library and its drivers are not stable. Nor is there a plan to stabilize it at the moment.

Developers

The sources of the main library and drivers can be found at `src/egl/`.

Lifetime of Display Resources

Contexts and surfaces are examples of display resources. They might live longer than the display that creates them.

In EGL, when a display is terminated through `eglTerminate`, all display resources should be destroyed. Similarly, when a thread is released through `eglReleaseThread`, all current display resources should be released. Another way to destroy or release resources is through functions such as `eglDestroySurface` or `eglMakeCurrent`.

When a resource that is current to some thread is destroyed, the resource should not be destroyed immediately. EGL requires the resource to live until it is no longer current. A driver usually calls `eglIs<Resource>Bound` to check if a resource is bound (current) to any thread in the destroy callbacks. If it is still bound, the resource is not destroyed.

The main library will mark destroyed current resources as unlinked. In a driver's `MakeCurrent` callback, `eglIs<Resource>Linked` can then be called to check if a newly released resource is linked to a display. If it is not, the last reference to the resource is removed and the driver should destroy the resource. But it should be careful here because `MakeCurrent` might be called with an uninitialized display.

This is the only mechanism provided by the main library to help manage the resources. The drivers are responsible to the correct behavior as defined by EGL.

EGL_RENDER_BUFFER

In EGL, the color buffer a context should try to render to is decided by the binding surface. It should try to render to the front buffer if the binding surface has `EGL_RENDER_BUFFER` set to `EGL_SINGLE_BUFFER`; If the same context is later bound to a surface with `EGL_RENDER_BUFFER` set to `EGL_BACK_BUFFER`, the context should try to render to the back buffer. However, the context is allowed to make the final decision as to which color buffer it wants to or is able to render to.

For pbuffer surfaces, the render buffer is always `EGL_BACK_BUFFER`. And for pixmap surfaces, the render buffer is always `EGL_SINGLE_BUFFER`. Unlike window surfaces, EGL spec requires their `EGL_RENDER_BUFFER` values to be honored. As a result, a driver should never set `EGL_PIXMAP_BIT` or `EGL_PBUFFER_BIT` bits of a config if the contexts created with the config won't be able to honor the `EGL_RENDER_BUFFER` of pixmap or pbuffer surfaces.

It should also be noted that pixmap and pbuffer surfaces are assumed to be single-buffered, in that `eglSwapBuffers` has no effect on them. It is desirable that a driver allocates a private color buffer for each pbuffer surface created. If the window system the driver supports has native pbuffers, or if the native pixmaps have more than one color buffers, the driver should carefully attach the native color buffers to the EGL surfaces, re-route them if required.

There is no defined behavior as to, for example, how `glDrawBuffer` interacts with `EGL_RENDER_BUFFER`. Right now, it is desired that the draw buffer in a client API be fixed for pixmap and pbuffer surfaces. Therefore, the driver is responsible to guarantee that the client API renders to the specified render buffer for pixmap and pbuffer surfaces.

`EGLDisplay` Mutex

The `EGLDisplay` will be locked before calling any of the dispatch functions (well, except for GetProcAddress which does not take an `EGLDisplay`). This guarantees that the same dispatch function will not be called with the sample

display at the same time. If a driver has access to an `EGLDisplay` without going through the EGL APIs, the driver should as well lock the display before using it.

CHAPTER 18

OpenGL ES

Mesa implements OpenGL ES 1.1 and OpenGL ES 2.0. More information about OpenGL ES can be found at http://www.khronos.org/opengles/.

OpenGL ES depends on a working EGL implementation. Please refer to Mesa EGL for more information about EGL.

Build the Libraries

1. Run `configure` with `--enable-gles1 --enable-gles2` and enable the Gallium driver for your hardware.
2. Build and install Mesa as usual.

Alternatively, if XCB-DRI2 is installed on the system, one can use `egl_dri2` EGL driver with OpenGL|ES-enabled DRI drivers

1. Run `configure` with `--enable-gles1 --enable-gles2`.
2. Build and install Mesa as usual.

Both methods will install libGLESv1_CM, libGLESv2, libEGL, and one or more EGL drivers for your hardware.

Run the Demos

There are some demos in `mesa/demos` repository.

Developers

Dispatch Table

OpenGL ES has an additional indirection when dispatching functions

```
Mesa:       glFoo() --> _mesa_Foo()
OpenGL ES:  glFoo() --> _es_Foo() --> _mesa_Foo()
```

The indirection serves several purposes

- When a function is in Mesa and the type matches, it checks the arguments and calls the Mesa function.
- When a function is in Mesa but the type mismatches, it checks and converts the arguments before calling the Mesa function.

- When a function is not available in Mesa, or accepts arguments that are not available in OpenGL, it provides its own implementation.

Other than the last case, OpenGL ES uses `APIspec.xml` to generate functions to check and/or converts the arguments.

CHAPTER 19

Environment Variables

Normally, no environment variables need to be set. Most of the environment variables used by Mesa/Gallium are for debugging purposes, but they can sometimes be useful for debugging end-user issues.

LibGL environment variables

- LIBGL_DEBUG - If defined debug information will be printed to stderr. If set to 'verbose' additional information will be printed.
- LIBGL_DRIVERS_PATH - colon-separated list of paths to search for DRI drivers
- LIBGL_ALWAYS_INDIRECT - forces an indirect rendering context/connection.
- LIBGL_ALWAYS_SOFTWARE - if set, always use software rendering
- LIBGL_NO_DRAWARRAYS - if set do not use DrawArrays GLX protocol (for debugging)
- LIBGL_SHOW_FPS - print framerate to stdout based on the number of glXSwapBuffers calls per second.
- LIBGL_DRI3_DISABLE - disable DRI3 if set (the value does not matter)

Core Mesa environment variables

- MESA_NO_ASM - if set, disables all assembly language optimizations
- MESA_NO_MMX - if set, disables Intel MMX optimizations
- MESA_NO_3DNOW - if set, disables AMD 3DNow! optimizations
- MESA_NO_SSE - if set, disables Intel SSE optimizations
- MESA_DEBUG - if set, error messages are printed to stderr. For example, if the application generates a GL_INVALID_ENUM error, a corresponding error message indicating where the error occurred, and possibly why, will be printed to stderr. For release builds, MESA_DEBUG defaults to off (no debug output). MESA_DEBUG accepts the following comma-separated list of named flags, which adds extra behaviour to just set MESA_DEBUG=1:
  - silent - turn off debug messages. Only useful for debug builds.
  - flush - flush after each drawing command
  - incomplete_tex - extra debug messages when a texture is incomplete
  - incomplete_fbo - extra debug messages when a fbo is incomplete
- MESA_LOG_FILE - specifies a file name for logging all errors, warnings, etc., rather than stderr

- MESA_TEX_PROG - if set, implement conventional texture env modes with fragment programs (intended for developers only)
- MESA_TNL_PROG - if set, implement conventional vertex transformation operations with vertex programs (intended for developers only). Setting this variable automatically sets the MESA_TEX_PROG variable as well.
- MESA_EXTENSION_OVERRIDE - can be used to enable/disable extensions. A value such as "GL_EXT_foo -GL_EXT_bar" will enable the GL_EXT_foo extension and disable the GL_EXT_bar extension.
- MESA_EXTENSION_MAX_YEAR - The GL_EXTENSIONS string returned by Mesa is sorted by extension year. If this variable is set to year X, only extensions defined on or before year X will be reported. This is to work-around a bug in some games where the extension string is copied into a fixed-size buffer without truncating. If the extension string is too long, the buffer overrun can cause the game to crash. This is a work-around for that.
- MESA_GL_VERSION_OVERRIDE - changes the value returned by glGetString(GL_VERSION) and possibly the GL API type.
  - The format should be MAJOR.MINOR[FC]
  - FC is an optional suffix that indicates a forward compatible context. This is only valid for versions >= 3.0.
  - GL versions < 3.0 are set to a compatibility (non-Core) profile
  - GL versions = 3.0, see below
  - GL versions > 3.0 are set to a Core profile
  - Examples: 2.1, 3.0, 3.0FC, 3.1, 3.1FC
    - 2.1 - select a compatibility (non-Core) profile with GL version 2.1
    - 3.0 - select a compatibility (non-Core) profile with GL version 3.0
    - 3.0FC - select a Core+Forward Compatible profile with GL version 3.0
    - 3.1 - select a Core profile with GL version 3.1
    - 3.1FC - select a Core+Forward Compatible profile with GL version 3.1
  - Mesa may not really implement all the features of the given version. (for developers only)
- MESA_GLES_VERSION_OVERRIDE - changes the value returned by glGetString(GL_VERSION) for OpenGL ES.
  - The format should be MAJOR.MINOR
  - Examples: 2.0, 3.0, 3.1
  - Mesa may not really implement all the features of the given version. (for developers only)
- MESA_GLSL_VERSION_OVERRIDE - changes the value returned by glGetString(GL_SHADING_LANGUAGE_VERSION). Valid values are integers, such as "130". Mesa will not really implement all the features of the given language version if it's higher than what's normally reported. (for developers only)
- MESA_GLSL - shading language compiler options
- MESA_NO_MINMAX_CACHE - when set, the minmax index cache is globally disabled.

Mesa Xlib driver environment variables

The following are only applicable to the Mesa Xlib software driver. See the Xlib software driver page for details.

- MESA_RGB_VISUAL - specifies the X visual and depth for RGB mode
- MESA_CI_VISUAL - specifies the X visual and depth for CI mode
- MESA_BACK_BUFFER - specifies how to implement the back color buffer, either "pixmap" or "ximage"
- MESA_GAMMA - gamma correction coefficients for red, green, blue channels
- MESA_XSYNC - enable synchronous X behavior (for debugging only)
- MESA_GLX_FORCE_CI - if set, force GLX to treat 8bpp visuals as CI visuals
- MESA_GLX_FORCE_ALPHA - if set, forces RGB windows to have an alpha channel.
- MESA_GLX_DEPTH_BITS - specifies default number of bits for depth buffer.
- MESA_GLX_ALPHA_BITS - specifies default number of bits for alpha channel.

i945/i965 driver environment variables (non-Gallium)

- INTEL_NO_HW - if set to 1, prevents batches from being submitted to the hardware. This is useful for debugging hangs, etc.
- INTEL_DEBUG - a comma-separated list of named flags, which do various things:
  - tex - emit messages about textures.
  - state - emit messages about state flag tracking
  - blit - emit messages about blit operations
  - miptree - emit messages about miptrees
  - perf - emit messages about performance issues
  - perfmon - emit messages about AMD_performance_monitor
  - bat - emit batch information
  - pix - emit messages about pixel operations
  - buf - emit messages about buffer objects
  - fbo - emit messages about framebuffers
  - fs - dump shader assembly for fragment shaders
  - gs - dump shader assembly for geometry shaders
  - sync - after sending each batch, emit a message and wait for that batch to finish rendering
  - prim - emit messages about drawing primitives
  - vert - emit messages about vertex assembly
  - dri - emit messages about the DRI interface
  - sf - emit messages about the strips & fans unit (for old gens, includes the SF program)
  - stats - enable statistics counters. you probably actually want perfmon or intel_gpu_top instead.
  - urb - emit messages about URB setup
  - vs - dump shader assembly for vertex shaders
  - clip - emit messages about the clip unit (for old gens, includes the CLIP program)
  - aub - dump batches into an AUB trace for use with simulation tools

  - shader_time - record how much GPU time is spent in each shader
  - no16 - suppress generation of 16-wide fragment shaders. useful for debugging broken shaders
  - blorp - emit messages about the blorp operations (blits & clears)
  - nodualobj - suppress generation of dual-object geometry shader code
  - optimizer - dump shader assembly to files at each optimization pass and iteration that make progress
  - ann - annotate IR in assembly dumps
  - no8 - don't generate SIMD8 fragment shader
  - vec4 - force vec4 mode in vertex shader
  - spill_fs - force spilling of all registers in the scalar backend (useful to debug spilling code)
  - spill_vec4 - force spilling of all registers in the vec4 backend (useful to debug spilling code)
  - cs - dump shader assembly for compute shaders
  - hex - print instruction hex dump with the disassembly
  - nocompact - disable instruction compaction
  - tcs - dump shader assembly for tessellation control shaders
  - tes - dump shader assembly for tessellation evaluation shaders
  - l3 - emit messages about the new L3 state during transitions
  - do32 - generate compute shader SIMD32 programs even if workgroup size doesn't exceed the SIMD16 limit
  - norbc - disable single sampled render buffer compression

Radeon driver environment variables (radeon, r200, and r300g)

- RADEON_NO_TCL - if set, disable hardware-accelerated Transform/Clip/Lighting.

EGL environment variables

Mesa EGL supports different sets of environment variables. See the Mesa EGL page for the details.

Gallium environment variables

- GALLIUM_HUD - draws various information on the screen, like framerate, cpu load, driver statistics, performance counters, etc. Set GALLIUM_HUD=help and run e.g. glxgears for more info.
- GALLIUM_HUD_PERIOD - sets the hud update rate in seconds (float). Use zero to update every frame. The default period is 1/2 second.
- GALLIUM_HUD_VISIBLE - control default visibility, defaults to true.
- GALLIUM_HUD_TOGGLE_SIGNAL - toggle visibility via user specified signal. Especially useful to toggle hud at specific points of application and disable for unencumbered viewing the rest of the time. For example, set GALLIUM_HUD_VISIBLE to false and GALLIUM_HUD_SIGNAL_TOGGLE to 10 (SIGUSR1). Use kill -10 to toggle the hud as desired.
- GALLIUM_LOG_FILE - specifies a file for logging all errors, warnings, etc. rather than stderr.

- GALLIUM_PRINT_OPTIONS - if non-zero, print all the Gallium environment variables which are used, and their current values.
- GALLIUM_DUMP_CPU - if non-zero, print information about the CPU on start-up
- TGSI_PRINT_SANITY - if set, do extra sanity checking on TGSI shaders and print any errors to stderr.
- DRAW_FSE - ???
- DRAW_NO_FSE - ???
- DRAW_USE_LLVM - if set to zero, the draw module will not use LLVM to execute shaders, vertex fetch, etc.
- ST_DEBUG - controls debug output from the Mesa/Gallium state tracker. Setting to "tgsi", for example, will print all the TGSI shaders. See src/mesa/state_tracker/st_debug.c for other options.

Softpipe driver environment variables

- SOFTPIPE_DUMP_FS - if set, the softpipe driver will print fragment shaders to stderr
- SOFTPIPE_DUMP_GS - if set, the softpipe driver will print geometry shaders to stderr
- SOFTPIPE_NO_RAST - if set, rasterization is no-op'd. For profiling purposes.
- SOFTPIPE_USE_LLVM - if set, the softpipe driver will try to use LLVM JIT for vertex shading processing.

LLVMpipe driver environment variables

- LP_NO_RAST - if set LLVMpipe will no-op rasterization
- LP_DEBUG - a comma-separated list of debug options is accepted. See the source code for details.
- LP_PERF - a comma-separated list of options to selectively no-op various parts of the driver. See the source code for details.
- LP_NUM_THREADS - an integer indicating how many threads to use for rendering. Zero turns off threading completely. The default value is the number of CPU cores present.

VMware SVGA driver environment variables

- SVGA_FORCE_SWTNL - force use of software vertex transformation
- SVGA_NO_SWTNL - don't allow software vertex transformation fallbacks (will often result in incorrect rendering).
- SVGA_DEBUG - for dumping shaders, constant buffers, etc. See the code for details.
- See the driver code for other, lesser-used variables.

VA-API state tracker environment variables

- VAAPI_MPEG4_ENABLED - enable MPEG4 for VA-API, disabled by default.

VC4 driver environment variables

- VC4_DEBUG - a comma-separated list of named flags, which do various things:
  - cl - dump command list during creation
  - qpu - dump generated QPU instructions
  - qir - dump QPU IR during program compile
  - nir - dump NIR during program compile
  - tgsi - dump TGSI during program compile
  - shaderdb - dump program compile information for shader-db analysis
  - perf - print during performance-related events
  - norast - skip actual hardware execution of commands
  - always_flush - flush after each draw call
  - always_sync - wait for finish after each flush
  - dump - write a GPU command stream trace file (VC4 simulator only)

Other Gallium drivers have their own environment variables. These may change frequently so the source code should be consulted for details.

CHAPTER 20

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 any of three software renderers:

1. llvmpipe - this is the high-performance Gallium LLVM driver
2. softpipe - this it the reference Gallium software driver
3. swrast - this is the legacy Mesa software rasterizer

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

Building OSMesa

Configure and build Mesa with something like:

```
configure --enable-osmesa --disable-driglx-direct --disable-dri --with-gallium-drivers=swrast
make
```

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

When the build is complete you should find:

```
lib/libOSMesa.so  (swrast-based OSMesa)
lib/gallium/libOSMsea.so  (gallium-based OSMesa)
```

Set your LD_LIBRARY_PATH to point to one directory or the other to select the library you want to use.

When you link your application, link with -lOSMesa

CHAPTER 21

Debugging Tips

Normally Mesa (and OpenGL) records but does not notify the user of errors. It is up to the application to call `glGetError` to check for errors. Mesa supports an environment variable, MESA_DEBUG, to help with debugging. If MESA_DEBUG is defined, a message will be printed to stdout whenever an error occurs.

More extensive error checking is done when Mesa is compiled with the DEBUG symbol defined. You'll have to edit the Make-config file and add -DDEBUG to the CFLAGS line for your system configuration. You may also want to replace any optimization flags with the -g flag so you can use your debugger. After you've edited Make-config type 'make clean' before recompiling.

In your debugger you can set a breakpoint in _mesa_error() to trap Mesa errors.

There is a display list printing/debugging facility. See the end of src/dlist.c for details.

CHAPTER 22

Performance Tips

Performance tips for software rendering:

1. Turn off smooth shading when you don't need it (glShadeModel)
2. Turn off depth buffering when you don't need it.
3. Turn off dithering when not needed.
4. Use double buffering as it's often faster than single buffering
5. Compile in the X Shared Memory extension option if it's supported on your system by adding -DSHM to CFLAGS and -lXext to XLIBS for your system in the Make-config file.
6. Recompile Mesa with more optimization if possible.
7. Try to maximize the amount of drawing done between glBegin/glEnd pairs.
8. Use the MESA_BACK_BUFFER variable to find best performance in double buffered mode. (X users only)
9. Optimized polygon rasterizers are employed when: rendering into back buffer which is an XImage RGB mode, not grayscale, not monochrome depth buffering is GL_LESS, or disabled flat or smooth shading dithered or non-dithered no other rasterization operations enabled (blending, stencil, etc)
10. Optimized line drawing is employed when: rendering into back buffer which is an XImage RGB mode, not grayscale, not monochrome depth buffering is GL_LESS or disabled flat shading dithered or non-dithered no other rasterization operations enabled (blending, stencil, etc)
11. Textured polygons are fastest when: using a 3-component (RGB), 2-D texture minification and magnification filters are GL_NEAREST texture coordinate wrap modes for S and T are GL_REPEAT GL_DECAL environment mode glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_FASTEST) depth buffering is GL_LESS or disabled
12. Lighting is fastest when: Two-sided lighting is disabled GL_LIGHT_MODEL_LOCAL_VIEWER is false GL_COLOR_MATERIAL is disabled No spot lights are used (all GL_SPOT_CUTOFFs are 180.0) No local lights are used (all position W's are 0.0) All material and light coefficients are >= zero
13. XFree86 users: if you want to use 24-bit color try starting your X server in 32-bit per pixel mode for better performance. That is, start your X server with startx – -bpp 32 instead of startx – -bpp 24
14. Try disabling dithering with the MESA_NO_DITHER environment variable. If this env var is defined Mesa will disable dithering and the command glEnable(GL_DITHER) will be ignored.

CHAPTER 23

Mesa Extensions

A number of extensions have been developed especially for Mesa. The specifications follow.

- MESA_agp_offset.spec
- MESA_copy_sub_buffer.spec
- MESA_drm_image.spec
- MESA_multithread_makecurrent.spec
- MESA_packed_depth_stencil.spec (obsolete)
- MESA_pack_invert.spec
- MESA_pixmap_colormap.spec
- MESA_program_debug.spec (obsolete)
- MESA_release_buffers.spec
- MESA_resize_buffers.spec (obsolete)
- MESA_set_3dfx_mode.spec
- MESA_shader_debug.spec
- MESA_sprite_point.spec (obsolete)
- MESA_swap_control.spec
- MESA_swap_frame_usage.spec
- MESA_texture_array.spec
- MESA_texture_signed_rgba.spec
- MESA_trace.spec (obsolete)
- MESA_window_pos.spec
- MESA_ycbcr_texture.spec
- WL_bind_wayland_display.spec

CHAPTER 24

Function Name Mangling

If you want to use both Mesa and another OpenGL library in the same application at the same time you may find it useful to compile Mesa with *name mangling*. This results in all the Mesa functions being prefixed with **mgl** instead of **gl**.

To do this, recompile Mesa with the compiler flag -DUSE_MGL_NAMESPACE. Add the flag to CFLAGS in the configuration file which you want to use. For example:

```
CFLAGS += -DUSE_MGL_NAMESPACE
```

CHAPTER 25

llvmpipe

The Gallium llvmpipe driver is a software rasterizer that uses LLVM to do runtime code generation. Shaders, point/line/triangle rasterization and vertex processing are implemented with LLVM IR which is translated to x86 or x86-64 machine code. Also, the driver is multithreaded to take advantage of multiple CPU cores (up to 8 at this time). It's the fastest software rasterizer for Mesa.

Requirements

- An x86 or amd64 processor; 64-bit mode recommended.

  Support for SSE2 is strongly encouraged. Support for SSSE3 and SSE4.1 will yield the most efficient code. The fewer features the CPU has the more likely is that you run into underperforming, buggy, or incomplete code.

  See /proc/cpuinfo to know what your CPU supports.

- LLVM: version 3.4 recommended; 3.3 or later required.

  For Linux, on a recent Debian based distribution do:

```
aptitude install llvm-dev
```

  For a RPM-based distribution do:

```
yum install llvm-devel
```

  For Windows you will need to build LLVM from source with MSVC or MINGW (either natively or through cross compilers) and CMake, and set the LLVM environment variable to the directory you installed it to. LLVM will be statically linked, so when building on MSVC it needs to be built with a matching CRT as Mesa, and you'll need to pass `-DLLVM_USE_CRT_xxx=yyy` as described below.

  LLVM build-type

  Mesa build-type

  debug,checked

  release,profile

  Debug

  `-DLLVM_USE_CRT_DEBUG=MTd`

  `-DLLVM_USE_CRT_DEBUG=MT`

  Release

  `-DLLVM_USE_CRT_RELEASE=MTd`

```
-DLLVM_USE_CRT_RELEASE=MT
```

You can build only the x86 target by passing -DLLVM_TARGETS_TO_BUILD=X86 to cmake.

- scons (optional)

Building

To build everything on Linux invoke scons as:

```
scons build=debug libgl-xlib
```

Alternatively, you can build it with GNU make, if you prefer, by invoking it as

```
make linux-llvm
```

but the rest of these instructions assume that scons is used. For Windows the procedure is similar except the target:

```
scons platform=windows build=debug libgl-gdi
```

Using

Linux

On Linux, building will create a drop-in alternative for libGL.so into

```
build/foo/gallium/targets/libgl-xlib/libGL.so
```

or:

```
lib/gallium/libGL.so
```

To use it set the LD_LIBRARY_PATH environment variable accordingly.

For performance evaluation pass build=release to scons, and use the corresponding lib directory without the "-debug" suffix.

Windows

On Windows, building will create `build/windows-x86-debug/gallium/targets/libgl-gdi/opengl32.dll` which is a drop-in alternative for system's `opengl32.dll`. To use it put it in the same directory as your application. It can also be used by replacing the native ICD driver, but it's quite an advanced usage, so if you need to ask, don't even try it.

There is however an easy way to replace the OpenGL software renderer that comes with Microsoft Windows 7 (or later) with llvmpipe (that is, on systems without any OpenGL drivers):

- copy build/windows-x86-debug/gallium/targets/libgl-gdi/opengl32.dll to C:\Windows\SysWOW64\mesadrv.dll
- load this registry settings:

```
REGEDIT4

; http://technet.microsoft.com/en-us/library/cc749368.aspx
; http://www.msfn.org/board/topic/143241-portable-windows-7-build-from-winpe-30/page-5#entry9425
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\OpenGLDrivers\MSOGL
"DLL"="mesadrv.dll"
```

```
    "DriverVersion"=dword:00000001
    "Flags"=dword:00000001
    "Version"=dword:00000002
```

- Ditto for 64 bits drivers if you need them.

Profiling

To profile llvmpipe you should build as

```
scons build=profile <same-as-before>
```

This will ensure that frame pointers are used both in C and JIT functions, and that no tail call optimizations are done by gcc.

Linux perf integration

On Linux, it is possible to have symbol resolution of JIT code with Linux perf:

```
perf record -g /my/application
perf report
```

When run inside Linux perf, llvmpipe will create a /tmp/perf-XXXXX.map file with symbol address table. It also dumps assembly code to /tmp/perf-XXXXX.map.asm, which can be used by the bin/perf-annotate-jit script to produce disassembly of the generated code annotated with the samples.

You can obtain a call graph via Gprof2Dot.

Unit testing

Building will also create several unit tests in build/linux-???-debug/gallium/drivers/llvmpipe:

- lp_test_blend: blending
- lp_test_conv: SIMD vector conversion
- lp_test_format: pixel unpacking/packing

Some of this tests can output results and benchmarks to a tab-separated-file for posterior analysis, e.g.:

```
build/linux-x86_64-debug/gallium/drivers/llvmpipe/lp_test_blend -o blend.tsv
```

Development Notes

- When looking to this code by the first time start in lp_state_fs.c, and then skim through the lp_bld_* functions called in there, and the comments at the top of the lp_bld_*.c functions.
- The driver-independent parts of the LLVM / Gallium code are found in src/gallium/auxiliary/gallivm/. The filenames and function prefixes need to be renamed from "lp_bld_" to something else though.
- We use LLVM-C bindings for now. They are not documented, but follow the C++ interfaces very closely, and appear to be complete enough for code generation. See this stand-alone example. See the llvm-c/Core.h file for reference.

Recommended Reading

- Rasterization
  - Triangle Scan Conversion using 2D Homogeneous Coordinates
  - Rasterization on Larrabee (DevMaster copy)
  - Rasterization using half-space functions
  - Advanced Rasterization
  - Optimizing Software Occlusion Culling
- Texture sampling
  - Perspective Texture Mapping
  - Texturing As In Unreal
  - Run-Time MIP-Map Filtering
  - Will "brilinear" filtering persist?
  - Trilinear filtering
  - Texture Swizzling
- SIMD
  - Whole-Function Vectorization
- Optimization
  - Optimizing Pixomatic For Modern x86 Processors
  - Intel 64 and IA-32 Architectures Optimization Reference Manual
  - Software optimization resources
  - Intel Intrinsics Guide
  -
- LLVM
  - LLVM Language Reference Manual
  - The secret of LLVM C bindings
- General
  - A trip through the Graphics Pipeline
  - WARP Architecture and Performance

CHAPTER 26

VMware guest GL driver

This page describes how to build, install and use the VMware guest GL driver (aka the SVGA or SVGA3D driver) for Linux using the latest source code. This driver gives a Linux virtual machine access to the host's GPU for hardware-accelerated 3D. VMware Workstation running on Linux or Windows and VMware Fusion running on MacOS are all supported.

With the August 2015 Workstation 12 / Fusion 8 releases, OpenGL 3.3 is supported in the guest. This requires:

- The VM is configured for virtual hardware version 12.
- The host OS, GPU and graphics driver supports DX11 (Windows) or OpenGL 4.0 (Linux, Mac)
- On Linux, the vmwgfx kernel module must be version 2.9.0 or later.
- A recent version of Mesa with the updated svga gallium driver.

Otherwise, OpenGL 2.1 is supported.

OpenGL 3.3 support can be disabled by setting the environment variable SVGA_VGPU10=0. You will then have OpenGL 2.1 support. This may be useful to work around application bugs (such as incorrect use of the OpenGL 3.x core profile).

Most modern Linux distros include the SVGA3D driver so end users shouldn't be concerned with this information. But if your distro lacks the driver or you want to update to the latest code these instructions explain what to do.

For more information about the X components see these wiki pages at x.org:

- Driver Overview
- xf86-video-vmware Details

Components

The components involved in this include:

- Linux kernel module: vmwgfx
- X server 2D driver: xf86-video-vmware
- User-space libdrm library
- Mesa/gallium OpenGL driver: "svga"

All of these components reside in the guest Linux virtual machine. On the host, all you're doing is running VMware Workstation or Fusion.

Prerequisites

- Kernel version at least 2.6.25
- Xserver version at least 1.7
- Ubuntu: For ubuntu you need to install a number of build dependencies.

```
sudo apt-get install git-core
sudo apt-get install automake libtool libpthread-stubs0-dev
sudo apt-get install xserver-xorg-dev x11proto-xinerama-dev libx11-xcb-dev
sudo apt-get install libxcb-glx0-dev libxrender-dev
sudo apt-get build-dep libgl1-mesa-dri libxcb-glx0-dev
```

- Fedora: For Fedora you also need to install a number of build dependencies.

```
sudo yum install mesa-libGL-devel xorg-x11-server-devel xorg-x11-util-macros
sudo yum install libXrender-devel.i686
sudo yum install automake gcc libtool expat-devel kernel-devel git-core
sudo yum install makedepend flex bison
```

Depending on your Linux distro, other packages may be needed. The configure scripts should tell you what's missing.

Getting the Latest Source Code

Begin by saving your current directory location:

```
export TOP=$PWD
```

- Mesa/Gallium master branch. This code is used to build libGL, and the direct rendering svga driver for libGL, vmwgfx_dri.so, and the X acceleration library libxatracker.so.x.x.x.

```
git clone git://anongit.freedesktop.org/git/mesa/mesa
```

- VMware Linux guest kernel module. Note that this repo contains the complete DRM and TTM code. The vmware-specific driver is really only the files prefixed with vmwgfx.

```
git clone git://anongit.freedesktop.org/git/mesa/vmwgfx
```

- libdrm, a user-space library that interfaces with drm. Most distros ship with this but it's safest to install a newer version. To get the latest code from git:

```
git clone git://anongit.freedesktop.org/git/mesa/drm
```

- xf86-video-vmware. The chainloading driver, vmware_drv.so, the legacy driver vmwlegacy_drv.so, and the vmwgfx driver vmwgfx_drv.so.

```
git clone git://anongit.freedesktop.org/git/xorg/driver/xf86-video-vmware
```

Building the Code

- Determine where the GL-related libraries reside on your system and set the LIBDIR environment variable accordingly.

  For 32-bit Ubuntu systems:

```
export LIBDIR=/usr/lib/i386-linux-gnu
```

For 64-bit Ubuntu systems:

```
export LIBDIR=/usr/lib/x86_64-linux-gnu
```

For 32-bit Fedora systems:

```
export LIBDIR=/usr/lib
```

For 64-bit Fedora systems:

```
export LIBDIR=/usr/lib64
```

- Build libdrm:

```
cd $TOP/drm
./autogen.sh --prefix=/usr --libdir=${LIBDIR}
make
sudo make install
```

- Build Mesa and the vmwgfx_dri.so driver, the vmwgfx_drv.so xorg driver, the X acceleration library libxatracker. The vmwgfx_dri.so is used by the OpenGL libraries during direct rendering, and by the Xorg server during accelerated indirect GL rendering. The libxatracker library is used exclusively by the X server to do render, copy and video acceleration:
  The following configure options doesn't build the EGL system.

```
cd $TOP/mesa
./autogen.sh --prefix=/usr --libdir=${LIBDIR} --with-gallium-drivers=svga --with-dri-drivers=swr
make
sudo make install
```

Note that you may have to install other packages that Mesa depends upon if they're not installed in your system. You should be told what's missing.

- xf86-video-vmware: Now, once libxatracker is installed, we proceed with building and replacing the current Xorg driver. First check if your system is 32- or 64-bit.

```
cd $TOP/xf86-video-vmware
./autogen.sh --prefix=/usr --libdir=${LIBDIR}
make
sudo make install
```

- vmwgfx kernel module. First make sure that any old version of this kernel module is removed from the system by issuing

```
sudo rm /lib/modules/`uname -r`/kernel/drivers/gpu/drm/vmwgfx.ko*
```

Build and install:

```
cd $TOP/vmwgfx
make
sudo make install
sudo depmod -a
```

If you're using a Ubuntu OS:

```
sudo update-initramfs -u
```

If you're using a Fedora OS:

```
sudo dracut --force
```

Add 'vmwgfx' to the /etc/modules file:

```
echo vmwgfx | sudo tee -a /etc/modules
```

Note: some distros put DRM kernel drivers in different directories. For example, sometimes vmwgfx.ko might be found in `/lib/modules/{version}/extra/vmwgfx.ko` or in `/lib/modules/{version}/kernel/drivers/gpu/drm/vmwgfx/vmwgfx.ko`.

After installing vmwgfx.ko you might want to run the following command to check that the new kernel module is in the expected place:

```
find /lib/modules -name vmwgfx.ko -exec ls -l '{}' \;
```

If you see the kernel module listed in more than one place, you may need to move things around.

Finally, if you update your kernel you'll probably have to rebuild and reinstall the vmwgfx.ko module again.

Now try to load the kernel module by issuing

```
sudo modprobe vmwgfx
```

Then type

```
dmesg
```

to watch the debug output. It should contain a number of lines prefixed with "[vmwgfx]". Then restart the Xserver (or reboot). The lines starting with "vmwlegacy" or "VMWARE" in the file /var/log/Xorg.0.log should now have been replaced with lines starting with "vmwgfx", indicating that the new Xorg driver is in use.

Running OpenGL Programs

In a shell, run 'glxinfo' and look for the following to verify that the driver is working:

```
OpenGL vendor string: VMware, Inc.
OpenGL renderer string: Gallium 0.4 on SVGA3D; build: RELEASE;
OpenGL version string: 2.1 Mesa 8.0
```

If you don't see this, try setting this environment variable:

```
export LIBGL_DEBUG=verbose
```

then rerun glxinfo and examine the output for error messages.

If OpenGL 3.3 is not working (you only get OpenGL 2.1):

Make sure the VM uses hardware version 12. Make sure the vmwgfx kernel module is version 2.9.0 or later. Check the vmware.log file for errors. Run 'dmesg | grep vmwgfx' and look for "DX: yes".

CHAPTER 27

Gallium Post-processing

The Gallium drivers support user-defined image post-processing. At the end of drawing a frame a post-processing filter can be applied to the rendered image. Example filters include morphological antialiasing and cell shading.

The filters can be toggled per-app via driconf, or per-session via the corresponding environment variables.

Multiple filters can be used together.

PP environment variables

- PP_DEBUG - If defined debug information will be printed to stderr.

Current filters

- pp_nored, pp_nogreen, pp_noblue - set to 1 to remove the corresponding color channel. These are basic filters for easy testing of the PP queue.
- pp_jimenezmlaa, pp_jimenezmlaa_color - Jimenez's MLAA is a morphological antialiasing filter. The two versions use depth and color data, respectively. Which works better depends on the app - depth will not blur text, but it will miss transparent textures for example. Set to a number from 2 to 32, roughly corresponding to quality. Numbers higher than 8 see minimizing gains.
- pp_celshade - set to 1 to enable cell shading (a more complex color filter).

CHAPTER 28

Application Issues

This page documents known issues with some OpenGL applications.

Topogun

Topogun for Linux (version 2, at least) creates a GLX visual without requesting a depth buffer. This causes bad rendering if the OpenGL driver happens to choose a visual without a depth buffer.

Mesa 9.1.2 and later (will) support a DRI configuration option to work around this issue. Using the driconf tool, set the "Create all visuals with a depth buffer" option before running Topogun. Then, all GLX visuals will be created with a depth buffer.

Old OpenGL games

Some old OpenGL games (approx. ten years or older) may crash during start-up because of an extension string buffer-overflow problem.

The problem is a modern OpenGL driver will return a very long string for the glGetString(GL_EXTENSIONS) query and if the application naively copies the string into a fixed-size buffer it can overflow the buffer and crash the application.

The work-around is to set the MESA_EXTENSION_MAX_YEAR environment variable to the approximate release year of the game. This will cause the glGetString(GL_EXTENSIONS) query to only report extensions older than the given year.

For example, if the game was released in 2001, do

```
export MESA_EXTENSION_MAX_YEAR=2001
```

before running the game. .. rubric:: Viewperf

name viewperf

See the Viewperf issues page for a detailed list of Viewperf issues.

CHAPTER 29

Viewperf Issues

This page lists known issues with SPEC Viewperf 11 and SPEC Viewperf 12 when running on Mesa-based drivers.

The Viewperf data sets are basically GL API traces that are recorded from CAD applications, then replayed in the Viewperf framework.

The primary problem with these traces is they blindly use features and OpenGL extensions that were supported by the OpenGL driver when the trace was recorded, but there's no checks to see if those features are supported by the driver when playing back the traces with Viewperf.

These issues have been reported to the SPEC organization in the hope that they'll be fixed in the future.

Viewperf 11

Some of the Viewperf 11 tests use a lot of memory. At least 2GB of RAM is recommended.

Catia-03 test 2

This test creates over 38000 vertex buffer objects. On some systems this can exceed the maximum number of buffer allocations. Mesa generates GL_OUT_OF_MEMORY errors in this situation, but Viewperf does no error checking and continues. When this happens, some drawing commands become no-ops. This can also eventually lead to a segfault either in Viewperf or the Mesa driver.

Catia-03 tests 3, 4, 8

These tests use features of the GL_NV_fragment_program2 and GL_NV_vertex_program3 extensions without checking if the driver supports them.

When Mesa tries to compile the vertex/fragment programs it generates errors (which Viewperf ignores). Subsequent drawing calls become no-ops and the rendering is incorrect.

sw-02 tests 1, 2, 4, 6

These tests depend on the GL_NV_primitive_restart extension.

If the Mesa driver doesn't support this extension the rendering will be incorrect and the test will fail.

Also, the color of the line drawings in test 2 seem to appear in a random color. This is probably due to some uninitialized state somewhere.

sw-02 test 6

The lines drawn in this test appear in a random color. That's because texture mapping is enabled when the lines are drawn, but no texture image is defined (glTexImage2D() is called with pixels=NULL). Since GL says the contents of the texture image are undefined in that situation, we get a random color.

Lightwave-01 test 3

This test uses a number of mipmapped textures, but the textures are incomplete because the last/smallest mipmap level (1 x 1 pixel) is never specified.

A trace captured with API trace shows this sequences of calls like this:

```
2504 glBindTexture(target = GL_TEXTURE_2D, texture = 55)
2505 glTexImage2D(target = GL_TEXTURE_2D, level = 0, internalformat = GL_RGBA, width = 512, height =
2506 glTexImage2D(target = GL_TEXTURE_2D, level = 1, internalformat = GL_RGBA, width = 256, height =
2507 glTexImage2D(target = GL_TEXTURE_2D, level = 2, internalformat = GL_RGBA, width = 128, height =
[...]
2512 glTexImage2D(target = GL_TEXTURE_2D, level = 7, internalformat = GL_RGBA, width = 4, height = 4,
2513 glTexImage2D(target = GL_TEXTURE_2D, level = 8, internalformat = GL_RGBA, width = 2, height = 2,
2514 glTexParameteri(target = GL_TEXTURE_2D, pname = GL_TEXTURE_MIN_FILTER, param = GL_LINEAR_MIPMAP_
2515 glTexParameteri(target = GL_TEXTURE_2D, pname = GL_TEXTURE_WRAP_S, param = GL_REPEAT)
2516 glTexParameteri(target = GL_TEXTURE_2D, pname = GL_TEXTURE_WRAP_T, param = GL_REPEAT)
2517 glTexParameteri(target = GL_TEXTURE_2D, pname = GL_TEXTURE_MAG_FILTER, param = GL_NEAREST)
```

Note that one would expect call 2514 to be glTexImage(level=9, width=1, height=1) but it's not there.

The minification filter is GL_LINEAR_MIPMAP_LINEAR and the texture's GL_TEXTURE_MAX_LEVEL is 1000 (the default) so a full mipmap is expected.

Later, these incomplete textures are bound before drawing calls. According to the GL specification, if a fragment program or fragment shader is being used, the sampler should return (0,0,0,1) ("black") when sampling from an incomplete texture. This is what Mesa does and the resulting rendering is darker than it should be.

It appears that NVIDIA's driver (and possibly AMD's driver) detects this case and returns (1,1,1,1) (white) which causes the rendering to appear brighter and match the reference image (however, AMD's rendering is *much* brighter than NVIDIA's).

If the fallback texture created in _mesa_get_fallback_texture() is initialized to be full white instead of full black the rendering appears correct. However, we have no plans to implement this work-around in Mesa.

Maya-03 test 2

This test makes some unusual calls to glRotate. For example:

```
glRotate(50, 50, 50, 1);
glRotate(100, 100, 100, 1);
glRotate(52, 52, 52, 1);
```

These unusual values lead to invalid modelview matrices. For example, the last glRotate command above produces this matrix with Mesa:

```
1.08536e+24 2.55321e-23 -0.000160389 0
5.96937e-25 1.08536e+24 103408 0
103408 -0.000160389 1.74755e+09 0
0 0 0 nan
```

and with NVIDIA's OpenGL:

```
1.4013e-45 0 -nan 0
0 1.4013e-45 1.4013e-45 0
1.4013e-45 -nan 1.4013e-45 0
0 0 0 1.4013e-45
```

This causes the object in question to be drawn in a strange orientation and with a semi-random color (between white and black) since GL_FOG is enabled.

Proe-05 test 1

This uses depth testing but there's two problems:

1. The glXChooseFBConfig() call doesn't request a depth buffer
2. The test never calls glClear(GL_DEPTH_BUFFER_BIT) to initialize the depth buffer

If the chosen visual does not have a depth buffer, you'll see the wireframe car model but it won't be rendered correctly.

If (by luck) the chosen visual has a depth buffer, its initial contents will be undefined so you may or may not see parts of the model. Interestingly, with NVIDIA's driver most visuals happen to have a depth buffer and apparently the contents are initialized to 1.0 by default so this test just happens to work with their drivers.

Finally, even if a depth buffer was requested and the glClear(GL_COLOR_BUFFER_BIT) calls were changed to glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) the problem still wouldn't be fixed because GL_DEPTH_WRITEMASK=GL_FALSE when glClear is called so clearing the depth buffer would be a no-op anyway.

Proe-05 test 6

This test draws an engine model with a two-pass algorithm. The first pass is drawn with polygon stipple enabled. The second pass is drawn without polygon stipple but with blending and GL_DEPTH_FUNC=GL_LEQUAL. If either of the two passes happen to use a software fallback of some sort, the Z values of fragments may be different between the two passes. This leads to incorrect rendering.

For example, the VMware SVGA gallium driver uses a special semi-fallback path for drawing with polygon stipple. Since the two passes are rendered with different vertex transformation implementations, the rendering doesn't appear as expected. Setting the SVGA_FORCE_SWTNL environment variable to 1 will force the driver to use the software vertex path all the time and clears up this issue.

According to the OpenGL invariance rules, there's no guarantee that the pixels produced by these two rendering states will match. To achieve invariance, both passes should enable polygon stipple and blending with appropriate patterns/modes to ensure the same fragments are produced in both passes.

Viewperf 12

Note that Viewperf 12 only runs on 64-bit Windows 7 or later.

catia-04

One of the catia tests calls wglGetProcAddress() to get some GL_EXT_direct_state_access functions (such as glBindMultiTextureEXT) and some GL_NV_half_float functions (such as glMultiTexCoord3hNV). If the extension/function is not supported, wglGetProcAddress() can return NULL. Unfortunately, Viewperf doesn't check for null pointers and crashes when it later tries to use the pointer.

Another catia test uses OpenGL 3.1's primitive restart feature. But when Viewperf creates an OpenGL context, it doesn't request version 3.1 If the driver returns version 3.0 or earlier all the calls related to primitive restart generate an OpenGL error. Some of the rendering is then incorrect.

energy-01

This test creates a 3D luminance texture of size 1K x 1K x 1K. If the OpenGL driver/device doesn't support a texture of this size the glTexImage3D() call will fail with GL_INVALID_VALUE or GL_OUT_OF_MEMORY and all that's rendered is plain white polygons. Ideally, the test would use a proxy texture to determine the max 3D texture size. But it does not do that.

maya-04

This test generates many GL_INVALID_OPERATION errors in its calls to glUniform(). Causes include:

- Trying to set float uniforms with glUniformi()
- Trying to set float uniforms with glUniform3f()
- Trying to set matrix uniforms with glUniform() instead of glUniformMatrix().

Apparently, the indexes returned by glGetUniformLocation() were hard-coded into the application trace when it was created. Since different implementations of glGetUniformLocation() may return different values for any given uniform name, subsequent calls to glUniform() will be invalid since they refer to the wrong uniform variables. This causes many OpenGL errors and leads to incorrect rendering.

medical-01

This test uses a single GLSL fragment shader which contains a GLSL 1.20 array initializer statement, but it neglects to specify `#version 120` at the top of the shader code. So, the shader does not compile and all that's rendered is plain white polygons.

Also, the test tries to create a very large 3D texture that may exceed the device driver's limit. When this happens, the glTexImage3D call fails and all that's rendered is a white box.

showcase-01

This is actually a DX11 test based on Autodesk's Showcase product. As such, it won't run with Mesa.

CHAPTER 30

Code Repository

Mesa uses git as its source code management system.

The master git repository is hosted on freedesktop.org.

You may access the repository either as an *anonymous user* (read-only) or as a *developer* (read/write).

You may also browse the main Mesa git repository and the Mesa demos and tests git repository.

Anonymous git Access

To get the Mesa sources anonymously (read-only):

1. Install the git software on your computer if needed.
2. Get an initial, local copy of the repository with:

```
git clone git://anongit.freedesktop.org/git/mesa/mesa
```

3. Later, you can update your tree from the master repository with:

```
git pull origin
```

4. If you also want the Mesa demos/tests repository:

```
git clone git://anongit.freedesktop.org/git/mesa/demos
```

Developer git Access

If you wish to become a Mesa developer with git-write privilege, please follow this procedure:

1. Subscribe to the mesa-dev mailing list.
2. Start contributing to the project by posting patches / review requests to the mesa-dev list. Specifically,
   - Use `git send-mail` to post your patches to mesa-dev.
   - Wait for someone to review the code and give you a `Reviewed-by` statement.
   - You'll have to rely on another Mesa developer to push your initial patches after they've been reviewed.
3. After you've demonstrated the ability to write good code and have had a dozen or so patches accepted you can apply for an account.

4. Occasionally, but rarely, someone may be given a git account sooner, but only if they're being supervised by another Mesa developer at the same organization and planning to work in a limited area of the code or on a separate branch.
5. To apply for an account, follow these directions. It's also appreciated if you briefly describe what you intend to do (work on a particular driver, add a new extension, etc.) in the bugzilla record.

Once your account is established:

1. Get an initial, local copy of the repository with:

```
git clone git+ssh://username@git.freedesktop.org/git/mesa/mesa
```

   Replace *username* with your actual login name.

2. Later, you can update your tree from the master repository with:

```
git pull origin
```

3. If you also want the Mesa demos/tests repository:

```
git clone git+ssh://username@git.freedesktop.org/git/mesa/demos
```

Windows Users

If you're using git on Windows you'll want to enable automatic CR/LF conversion in your local copy of the repository:

```
git config --global core.autocrlf true
```

This will cause git to convert all text files to CR+LF on checkout, and to LF on commit.

Unix users don't need to set this option.

Development Branches

At any given time, there may be several active branches in Mesa's repository. Generally, the trunk contains the latest development (unstable) code while a branch has the latest stable code.

The command `git-branch` will list all available branches.

Questions about branch status/activity should be posted to the mesa3d-dev mailing list.

Developer Git Tips

1. Setting up to edit the master branch

   If you try to do a pull by just saying‘‘ git pull ‘‘ and git complains that you have not specified a branch, try:

```
git config branch.master.remote origin
git config branch.master.merge master
```

   Otherwise, you have to say‘‘ git pull origin master ‘‘ each time you do a pull.

2. Small changes to master

   If you are an experienced git user working on substantial modifications, you are probably working on a separate branch and would rebase your branch prior to merging with master. But for small changes to the master branch itself, you also need to use the rebase feature in order to avoid an unnecessary and distracting branch in master.

   If it has been awhile since you've done the initial clone, try

```
git pull
```

   to get the latest files before you start working.

   Make your changes and use

```
git add <files to commit>
git commit
```

   to get your changes ready to push back into the fd.o repository.

   It is possible (and likely) that someone has changed master since you did your last pull. Even if your changes do not conflict with their changes, git will make a fast-forward merge branch, branching from the point in time where you did your last pull and merging it to a point after the other changes.

   To avoid this,

```
git pull --rebase
git push
```

   If you are familiar with CVS or similar system, this is similar to doing a '' cvs update '' in order to update your source tree to the current repository state, instead of the time you did the last update. (CVS doesn't work like git in this respect, but this is easiest way to explain it.)
   In any case, your repository now looks like you made your changes after all the other changes.

   If the rebase resulted in conflicts or changes that could affect the proper operation of your changes, you'll need to investigate those before doing the push.

   If you want the rebase action to be the default action, then

```
git config branch.master.rebase true
git config --global branch.autosetuprebase=always
```

   See Understanding Git Conceptually for a fairly clear explanation about all of this.

CHAPTER 31

Mesa source code tree overview

This is a brief summary of Mesa's directory tree and what's contained in each directory.

- **docs** - Documentation
- **include** - Public OpenGL header files
- **src**
  - **egl** - EGL library sources
    - **docs** - EGL documentation
    - **drivers** - EGL drivers
    - **main** - main EGL library implementation. This is where all the EGL API functions are implemented, like eglCreateContext().
  - **glsl** - the GLSL compiler
  - **mapi** - Mesa APIs
  - **glapi** - OpenGL API dispatch layer. This is where all the GL entrypoints like glClear, glBegin, etc. are generated, as well as the GL dispatch table. All GL function calls jump through the dispatch table to functions found in main/.
  - **mesa** - Main Mesa sources
    - **main** - The core Mesa code (mainly state management)
    - **drivers** - Mesa drivers (not used with Gallium)
      - **common** - code which may be shared by all drivers
      - **dri** - Direct Rendering Infrastructure drivers
      - **common** - code shared by all DRI drivers
      - **i915** - driver for Intel i915/i945
      - **i965** - driver for Intel i965
      - **radeon** - driver for ATI R100
      - **r200** - driver for ATI R200
      - XXX more
      - **x11** - Xlib-based software driver
      - **osmesa** - off-screen software driver

        - XXX more
    - **math** - vertex array translation and transformation code (not used with Gallium)
    - **program** - Vertex/fragment shader and GLSL compiler code
    - **sparc** - Assembly code/optimizations for SPARC systems (not used with Gallium)
    - **state_tracker** - State tracker / driver for Gallium. This is basically a Mesa device driver that speaks to Gallium. This directory may be moved to src/mesa/drivers/gallium at some point.
    - **swrast** - Software rasterization module. For drawing points, lines, triangles, bitmaps, images, etc. in software. (not used with Gallium)
    - **swrast_setup** - Software primitive setup. Does things like polygon culling, glPolygonMode, polygon offset, etc. (not used with Gallium)
    - **tnl** - Software vertex Transformation 'n Lighting. (not used with Gallium)
    - **tnl_dd** - TNL code for device drivers. (not used with Gallium)
    - **vbo** - Vertex Buffer Object code. All drawing with glBegin/glEnd, glDrawArrays, display lists, etc. goes through this module. The results is a well-defined set of vertex arrays which are passed to the device driver (or tnl module) for rendering.
    - **x86** - Assembly code/optimizations for 32-bit x86 systems (not used with Gallium)
    - **x86-64** - Assembly code/optimizations for 64-bit x86 systems (not used with Gallium)
- **gallium** - Gallium3D source code
    - **include** - Gallium3D header files which define the Gallium3D interfaces
    - **drivers** - Gallium3D device drivers
        - **i915** - Driver for Intel i915/i945.
        - **llvmpipe** - Software driver using LLVM for runtime code generation.
        - **nv*** - Drivers for NVIDIA GPUs.
        - **radeonsi** - Driver for AMD Southern Island.
        - **r300** - Driver for ATI R300 - R500.
        - **r600** - Driver for ATI/AMD R600 - Northern Island.
        - **softpipe** - Software reference driver.
        - **svga** - Driver for VMware's SVGA virtual GPU.
        - **trace** - Driver for tracing Gallium calls.
        - XXX more
    - **auxiliary** - Gallium support code
        - **draw** - Software vertex processing and primitive assembly module. This includes vertex program execution, clipping, culling and optional stages for drawing wide lines, stippled lines, polygon stippling, two-sided lighting, etc. Intended for use by drivers for hardware that does not have vertex shaders. Geometry shaders will also be implemented in this module.
        - **cso_cache** - Constant State Objects Cache. Used to filter out redundant state changes between state trackers and drivers.
        - **gallivm** - LLVM module for Gallium. For LLVM-based compilation, optimization and code generation for TGSI shaders. Incomplete.

        - **pipebuffer** - utility module for managing buffers
        - **rbug** - Gallium remote debug utility
        - **rtasm** - run-time assembly/machine code generation. Currently there's run-time code generation for x86/SSE, PowerPC and Cell SPU.
        - **tgsi** - TG Shader Infrastructure. Code for encoding, manipulating and interpreting GPU programs.
        - **translate** - module for translating vertex data from one format to another.
        - **util** - assorted utilities for arithmetic, hashing, surface creation, memory management, 2D blitting, simple rendering, etc.
      * **state_trackers** -
        - **clover** - OpenCL state tracker
        - **dri** - Meta state tracker for DRI drivers
        - **glx** - Meta state tracker for GLX
        - **vdpau** - VDPAU state tracker
        - **wgl** -
        - **xorg** - Meta state tracker for Xorg video drivers
        - **xvmc** - XvMC state tracker
      * **winsys** -
        - **drm** -
        - **gdi** -
        - **xlib** -
    - **glx** - The GLX library code for building libGL. This is used for direct rendering drivers. It will dynamically load one of the xxx_dri.so drivers.
- **lib** - where the GL libraries are placed

CHAPTER 32

Development Utilities

Mesa demos collection includes several utility routines in the `src/util/` directory.

Piglit is an open-source test suite for OpenGL implementations.

ApiTrace is a project to trace, analyze and debug graphics api's.

Valgrind is a very useful tool for tracking down memory-related problems in your code.

Coverity provides static code analysis of Mesa. If you create an account you can see the results and try to fix outstanding issues.

CHAPTER 33

Help Wanted / To-Do List

We can always use more help with the Mesa project. Here are some specific ideas and areas where help would be appreciated:

1. **Driver patching and testing.** Patches are often posted to the mesa-dev mailing list, but aren't immediately checked into git because not enough people are testing them. Just applying patches, testing and reporting back is helpful.
2. **Driver debugging.** There are plenty of open bugs in the bug database.
3. **Remove aliasing warnings.** Enable gcc -Wstrict-aliasing=2 -fstrict-aliasing and track down aliasing issues in the code.
4. **Windows driver building, testing and maintenance.** Fixing MSVC builds.
5. **Contribute more tests to 'Piglit <http://piglit.freedesktop.org/>'__.**
6. **Automatic testing.** It would be great if someone would set up an automated system for grabbing the latest Mesa code and run tests (such as piglit) then report issues to the mailing list.

You can find some further To-do lists here:

Common To-Do lists:

- **GL3.txt** - Status of OpenGL 3.x / 4.x features in Mesa.
- **MissingFunctionality** - Detailed information about missing OpenGL features.

Driver specific To-Do lists:

- **LLVMpipe** - Software driver using LLVM for runtime code generation.
- **radeonsi** - Driver for AMD Southern Island.
- **r600g** - Driver for ATI/AMD R600 - Northern Island.
- **r300g** - Driver for ATI R300 - R500.
- **i915g** - Driver for Intel i915/i945.

If you want to do something new in Mesa, first join the Mesa developer's mailing list. Then post a message to propose what you want to do, just to make sure there's no issues.

Anyone is welcome to contribute code to the Mesa project. By doing so, it's assumed that you agree to the code's licensing terms.

Finally:

1. Try to write high-quality code that follows the existing style.
2. Use uniform indentation, write comments, use meaningful identifiers, etc.

3. Test your code thoroughly. Include test programs if appropriate.

CHAPTER 34

Development Notes

- *Coding Style*
- *Submitting Patches*
- *Making a New Mesa Release*
- *Adding Extensions*

Coding Style

Mesa is over 20 years old and the coding style has evolved over time. Some old parts use a style that's a bit out of date. If the guidelines below don't cover something, try following the format of existing, neighboring code.

Basic formatting guidelines

- 3-space indentation, no tabs.
- Limit lines to 78 or fewer characters. The idea is to prevent line wrapping in 80-column editors and terminals. There are exceptions, such as if you're defining a large, static table of information.
- Opening braces go on the same line as the if/for/while statement. For example:

```
if (condition) {
   foo;
} else {
   bar;
}
```

- Put a space before/after operators. For example, `a = b + c;` and not `a=b+c;`
- This GNU indent command generally does the right thing for formatting:

```
indent -br -i3 -npcs --no-tabs infile.c -o outfile.c
```

- Use comments wherever you think it would be helpful for other developers. Several specific cases and style examples follow. Note that we roughly follow Doxygen conventions.

  Single-line comments:

```
/* null-out pointer to prevent dangling reference below */
bufferObj = NULL;
```

  Or,

```
bufferObj = NULL;  /* prevent dangling reference below */
```

Multi-line comment:

```
/* If this is a new buffer object id, or one which was generated but
 * never used before, allocate a buffer object now.
 */
```

We try to quote the OpenGL specification where prudent:

```
/* Page 38 of the PDF of the OpenGL ES 3.0 spec says:
 *
 *     "An INVALID_OPERATION error is generated for any of the following
 *     conditions:
 *
 *     *  is zero."
 *
 * Additionally, page 94 of the PDF of the OpenGL 4.5 core spec
 * (30.10.2014) also says this, so it's no longer allowed for desktop GL,
 * either.
 */
```

Function comment example:

```
/**
 * Create and initialize a new buffer object.  Called via the
 * ctx->Driver.CreateObject() driver callback function.
 * \param  name  integer name of the object
 * \param  type  one of GL_FOO, GL_BAR, etc.
 * \return  pointer to new object or NULL if error
 */
struct gl_object *
_mesa_create_object(GLuint name, GLenum type)
{
   /* function body */
}
```

- Put the function return type and qualifiers on one line and the function name and parameters on the next, as seen above. This makes it easy to use `grep ^function_name dir/*` to find function definitions. Also, the opening brace goes on the next line by itself (see above.)
- Function names follow various conventions depending on the type of function:

```
glFooBar()        - a public GL entry point (in glapi_dispatch.c)
_mesa_FooBar()    - the internal immediate mode function
save_FooBar()     - retained mode (display list) function in dlist.c
foo_bar()         - a static (private) function
_mesa_foo_bar()   - an internal non-static Mesa function
```

- Constants, macros and enumerant names are ALL_UPPERCASE, with _ between words.
- Mesa usually uses camel case for local variables (Ex: “localVarname”) while gallium typically uses underscores (Ex: “local_var_name”).
- Global variables are almost never used because Mesa should be thread-safe.
- Booleans. Places that are not directly visible to the GL API should prefer the use of `bool`, `true`, and `false` over `GLboolean`, `GL_TRUE`, and `GL_FALSE`. In C code, this may mean that `#include <stdbool.h>` needs to be added. The `try_emit_*` methods in src/mesa/program/ir_to_mesa.cpp and src/mesa/state_tracker/st_glsl_to_tgsi.cpp can serve as examples.

Submitting patches

The basic guidelines for submitting patches are:

- Patches should be sufficiently tested before submitting.
- Code patches should follow Mesa coding conventions.
- Whenever possible, patches should only effect individual Mesa/Gallium components.
- Patches should never introduce build breaks and should be bisectable (see `git bisect`.)
- Patches should be properly formatted (see below).
- Patches should be submitted to mesa-dev for review using `git send-email`.
- Patches should not mix code changes with code formatting changes (except, perhaps, in very trivial cases.)

Patch formatting

The basic rules for patch formatting are:

- Lines should be limited to 75 characters or less so that git logs displayed in 80-column terminals avoid line wrapping. Note that git log uses 4 spaces of indentation (4 + 75 < 80).
- The first line should be a short, concise summary of the change prefixed with a module name. Examples:

```
mesa: Add support for querying GL_VERTEX_ATTRIB_ARRAY_LONG

gallium: add PIPE_CAP_DEVICE_RESET_STATUS_QUERY

i965: Fix missing type in local variable declaration.
```

- Subsequent patch comments should describe the change in more detail, if needed. For example:

```
i965: Remove end-of-thread SEND alignment code.

This was present in Eric's initial implementation of the compaction code
for Sandybridge (commit 077d01b6). There is no documentation saying this
is necessary, and removing it causes no regressions in piglit on any
platform.
```

- A "Signed-off-by:" line is not required, but not discouraged either.
- If a patch address a bugzilla issue, that should be noted in the patch comment. For example:

```
Bugzilla: https://bugs.freedesktop.org/show_bug.cgi?id=89689
```

- If there have been several revisions to a patch during the review process, they should be noted such as in this example:

```
st/mesa: add ARB_texture_stencil8 support (v4)

if we support stencil texturing, enable texture_stencil8
there is no requirement to support native S8 for this,
the texture can be converted to x24s8 fine.

v2: fold fixes from Marek in:
   a) put S8 last in the list
   b) fix renderable to always test for d/s renderable
    fixup the texture case to use a stencil only format
```

```
        for picking the format for the texture view.
    v3: hit fallback for getteximage
    v4: put s8 back in front, it shouldn't get picked now (Ilia)
```

- If someone tested your patch, document it with a line like this:

```
    Tested-by: Joe Hacker <jhacker@foo.com>
```

- If the patch was reviewed (usually the case) or acked by someone, that should be documented with:

```
    Reviewed-by: Joe Hacker <jhacker@foo.com>
    Acked-by: Joe Hacker <jhacker@foo.com>
```

Testing Patches

It should go without saying that patches must be tested. In general, do whatever testing is prudent.

You should always run the Mesa test suite before submitting patches. The test suite can be run using the 'make check' command. All tests must pass before patches will be accepted, this may mean you have to update the tests themselves.

Whenever possible and applicable, test the patch with Piglit to check for regressions.

Mailing Patches

Patches should be sent to the mesa-dev mailing list for review: mesa-dev@lists.freedesktop.org‘ <>‘__. When submitting a patch make sure to use git send-email rather than attaching patches to emails. Sending patches as attachments prevents people from being able to provide in-line review comments.

When submitting follow-up patches you can use –in-reply-to to make v2, v3, etc patches show up as replies to the originals. This usually works well when you're sending out updates to individual patches (as opposed to re-sending the whole series). Using –in-reply-to makes it harder for reviewers to accidentally review old patches.

When submitting follow-up patches you should also login to patchwork and change the state of your old patches to Superseded.

Reviewing Patches

When you've reviewed a patch on the mailing list, please be unambiguous about your review. That is, state either

```
Reviewed-by: Joe Hacker <jhacker@foo.com>
```

or:

```
Acked-by: Joe Hacker <jhacker@foo.com>
```

Rather than saying just "LGTM" or "Seems OK". If small changes are suggested, it's OK to say something like:

```
With the above fixes, Reviewed-by: Joe Hacker <jhacker@foo.com>
```

which tells the patch author that the patch can be committed, as long as the issues are resolved first. .. rubric:: Marking a commit as a candidate for a stable branch

name marking-a-commit-as-a-candidate-for-a-stable-branch

If you want a commit to be applied to a stable branch, you should add an appropriate note to the commit message.

Here are some examples of such a note:

- CC: <mesa-stable@lists.freedesktop.org>
- CC: "9.2 10.0" <mesa-stable@lists.freedesktop.org>
- CC: "10.0" <mesa-stable@lists.freedesktop.org>

Simply adding the CC to the mesa-stable list address is adequate to nominate the commit for the most-recently-created stable branch. It is only necessary to specify a specific branch name, (such as "9.2 10.0" or "10.0" in the examples above), if you want to nominate the commit for an older stable branch. And, as in these examples, you can nominate the commit for the older branch in addition to the more recent branch, or nominate the commit exclusively for the older branch. This "CC" syntax for patch nomination will cause patches to automatically be copied to the mesa-stable@ mailing list when you use "git send-email" to send patches to the mesa-dev@ mailing list. Also, if you realize that a commit should be nominated for the stable branch after it has already been committed, you can send a note directly to the mesa-stable@lists.freedesktop.org where the Mesa stable-branch maintainers will receive it. Be sure to mention the commit ID of the commit of interest (as it appears in the mesa master branch). The latest set of patches that have been nominated, accepted, or rejected for the upcoming stable release can always be seen on the Mesa Stable Queue page. .. rubric:: Criteria for accepting patches to the stable branch

name criteria-for-accepting-patches-to-the-stable-branch

Mesa has a designated release manager for each stable branch, and the release manager is the only developer that should be pushing changes to these branches. Everyone else should simply nominate patches using the mechanism described above. The stable-release manager will work with the list of nominated patches, and for each patch that meets the crtieria below will cherry-pick the patch with: `git cherry-pick -x <commit>`. The `-x` option is important so that the picked patch references the comit ID of the original patch. The stable-release manager may at times need to force-push changes to the stable branches, for example, to drop a previously-picked patch that was later identified as causing a regression). These force-pushes may cause changes to be lost from the stable branch if developers push things directly. Consider yourself warned. The stable-release manager is also given broad discretion in rejecting patches that have been nominated for the stable branch. The most basic rule is that the stable branch is for bug fixes only, (no new features, no regressions). Here is a non-exhaustive list of some reasons that a patch may be rejected:

- Patch introduces a regression. Any reported build breakage or other regression caused by a particular patch, (game no longer work, piglit test changes from PASS to FAIL), is justification for rejecting a patch.
- Patch is too large, (say, larger than 100 lines)
- Patch is not a fix. For example, a commit that moves code around with no functional change should be rejected.
- Patch fix is not clearly described. For example, a commit message of only a single line, no description of the bug, no mention of bugzilla, etc.
- Patch has not obviously been reviewed, For example, the commit message has no Reviewed-by, Signed-off-by, nor Tested-by tags from anyone but the author.
- Patch has not already been merged to the master branch. As a rule, bug fixes should never be applied first to a stable branch. Patches should land first on the master branch and then be cherry-picked to a stable branch. (This is to avoid future releases causing regressions if the patch is not also applied to master.) The only things that might look like exceptions would be backports of patches from master that happen to look significantly different.
- Patch depends on too many other patches. Ideally, all stable-branch patches should be self-contained. It sometimes occurs that a single, logical bug-fix occurs as two separate patches on master, (such as an original patch, then a subsequent fix-up to that patch). In such a case, these two patches should be squashed into a single, self-contained patch for the stable branch. (Of course, if the squashing makes the patch too large, then that could be a reason to reject the patch.)

- Patch includes new feature development, not bug fixes. New OpenGL features, extensions, etc. should be applied to Mesa master and included in the next major release. Stable releases are intended only for bug fixes. Note: As an exception to this rule, the stable-release manager may accept hardware-enabling "features". For example, backports of new code to support a newly-developed hardware product can be accepted if they can be reasonably determined to not have effects on other hardware.
- Patch is a performance optimization. As a rule, performance patches are not candidates for the stable branch. The only exception might be a case where an application's performance was recently severely impacted so as to become unusable. The fix for this performance regression could then be considered for a stable branch. The optimization must also be non-controversial and the patches still need to meet the other criteria of being simple and self-contained
- Patch introduces a new failure mode (such as an assert). While the new assert might technically be correct, for example to make Mesa more conformant, this is not the kind of "bug fix" we want in a stable release. The potential problem here is that an OpenGL program that was previously working, (even if technically non-compliant with the specification), could stop working after this patch. So that would be a regression that is unaacceptable for the stable branch.

Making a New Mesa Release

These are the instructions for making a new Mesa release.

Get latest source files

Use git to get the latest Mesa files from the git repository, from whatever branch is relevant. This document uses the convention X.Y.Z for the release being created, which should be created from a branch named X.Y.

Perform basic testing

The release manager should, at the very least, test the code by compiling it, installing it, and running the latest piglit to ensure that no piglit tests have regressed since the previous release.

The release manager should do this testing with at least one hardware driver, (say, whatever is contained in the local development machine), as well as on both Gallium and non-Gallium software drivers. The software testing can be performed by running piglit with the following environment-variable set:

```
LIBGL_ALWAYS_SOFTWARE=1
```

And Gallium vs. non-Gallium software drivers can be obtained by using the following configure flags on separate builds:

```
--with-dri-drivers=swrast
--with-gallium-drivers=swrast
```

Note: If both options are given in one build, both swrast_dri.so drivers will be compiled, but only one will be installed. The following command can be used to ensure the correct driver is being tested:

```
LIBGL_ALWAYS_SOFTWARE=1 glxinfo | grep "renderer string"
```

If any regressions are found in this testing with piglit, stop here, and do not perform a release until regressions are fixed. .. rubric:: Update version in file VERSION

name update-version-in-file-version

Increment the version contained in the file VERSION at Mesa's top-level, then commit this change.

Create release notes for the new release

Create a new file docs/relnotes/X.Y.Z.html, (follow the style of the previous release notes). Note that the sha256sums section of the release notes should be empty at this point.

Two scripts are available to help generate portions of the release notes:

```
./bin/bugzilla_mesa.sh
./bin/shortlog_mesa.sh
```

The first script identifies commits that reference bugzilla bugs and obtains the descriptions of those bugs from bugzilla. The second script generates a log of all commits. In both cases, HTML-formatted lists are printed to stdout to be included in the release notes.

Commit these changes

Make the release archives, signatures, and the release tag

From inside the Mesa directory:

```
./autogen.sh
make -j1 tarballs
```

After the tarballs are created, the sha256 checksums for the files will be computed and printed. These will be used in a step below.

It's important at this point to also verify that the constructed tar file actually builds:

```
tar xjf MesaLib-X.Y.Z.tar.bz2
cd Mesa-X.Y.Z
./configure --enable-gallium-llvm
make -j6
make install
```

Some touch testing should also be performed at this point, (run glxgears or more involved OpenGL programs against the installed Mesa).

Create detached GPG signatures for each of the archive files created above:

```
gpg --sign --detach MesaLib-X.Y.Z.tar.gz
gpg --sign --detach MesaLib-X.Y.Z.tar.bz2
gpg --sign --detach MesaLib-X.Y.Z.zip
```

Tag the commit used for the build:

```
git tag -s mesa-X.Y.X -m "Mesa X.Y.Z release"
```

Note: It would be nice to investigate and fix the issue that causes the tarballs target to fail with multiple build process, such as with "-j4". It would also be nice to incorporate all of the above commands into a single makefile target. And instead of a custom "tarballs" target, we should incorporate things into the standard "make dist" and "make distcheck" targets.

Add the sha256sums to the release notes

Edit docs/relnotes/X.Y.Z.html to add the sha256sums printed as part of "make tarballs" in the previous step. Commit this change.

Push all commits and the tag created above

This is the first step that cannot easily be undone. The release is going forward from this point:

```
git push origin X.Y --tags
```

Install the release files and signatures on the distribution server

The following commands can be used to copy the release archive files and signatures to the freedesktop.org server:

```
scp MesaLib-X.Y.Z* people.freedesktop.org:
ssh people.freedesktop.org
cd /srv/ftp.freedesktop.org/pub/mesa
mkdir X.Y.Z
cd X.Y.Z
mv ~/MesaLib-X.Y.Z* .
```

Back on mesa master, add the new release notes into the tree

Something like the following steps will do the trick:

```
cp docs/relnotes/X.Y.Z.html /tmp
    git checkout master
    cp /tmp/X.Y.Z.html docs/relnotes
    git add docs/relnotes/X.Y.Z.html
```

Also, edit docs/relnotes.html to add a link to the new release notes, and edit docs/index.html to add a news entry. Then commit and push:

```
git commit -a -m "docs: Import X.Y.Z release notes, add news item."
    git push origin
```

Update the mesa3d.org website

NOTE: The recent release managers have not been performing this step themselves, but leaving this to Brian Paul, (who has access to the sourceforge.net hosting for mesa3d.org). Brian is more than willing to grant the permission necessary to future release managers to do this step on their own.

Update the web site by copying the docs/ directory's files to /home/users/b/br/brianp/mesa-www/htdocs/ with:
‘‘ sftp USERNAME,mesa3d@web.sourceforge.net‘‘

Announce the release

Make an announcement on the mailing lists: *mesa-dev@lists.freedesktop.org*, and *mesa-announce@lists.freedesktop.org* Follow the template of previously-sent release announcements. The following command can be used to generate the log of changes to be included in the release announcement:

```
git shortlog mesa-X.Y.Z-1..mesa-X.Y.Z
```

Adding Extensions

To add a new GL extension to Mesa you have to do at least the following.

- If glext.h doesn't define the extension, edit include/GL/gl.h and add code like this:

```
#ifndef GL_EXT_the_extension_name
#define GL_EXT_the_extension_name 1
/* declare the new enum tokens */
/* prototype the new functions */
/* TYPEDEFS for the new functions */
#endif
```

- In the src/mapi/glapi/gen/ directory, add the new extension functions and enums to the gl_API.xml file. Then, a bunch of source files must be regenerated by executing the corresponding Python scripts.
- Add a new entry to the `gl_extensions` struct in mtypes.h if the extension requires driver capabilities not already exposed by another extension.
- Add a new entry to the src/mesa/main/extensions_table.h file.
- From this point, the best way to proceed is to find another extension, similar to the new one, that's already implemented in Mesa and use it as an example.
- If the new extension adds new GL state, the functions in get.c, enable.c and attrib.c will most likely require new code.
- To determine if the new extension is active in the current context, use the auto-generated _mesa_has_##name_str() function defined in src/mesa/main/extensions.h.
- The dispatch tests check_table.cpp and dispatch_sanity.cpp should be updated with details about the new extensions functions. These tests are run using 'make check'

CHAPTER 35

Source Code Documentation

Doxygen is used to automatically produce cross-referenced documentation from the Mesa source code.

The Doxygen configuration files and generated files are not included in the normal Mesa distribution (they're very large). To generate Doxygen documentation, download Mesa from git, change to the `doxygen` directory and run `make`.

For an example of Doxygen usage in Mesa, see a recent source file such as bufferobj.c.

If you're reading this page from your local copy of Mesa, and have run the doxygen scripts, you can read the documentation here

CHAPTER 36

GL Dispatch in Mesa

Several factors combine to make efficient dispatch of OpenGL functions fairly complicated. This document attempts to explain some of the issues and introduce the reader to Mesa's implementation. Readers already familiar with the issues around GL dispatch can safely skip ahead to the *overview of Mesa's implementation*.

1. Complexity of GL Dispatch

Every GL application has at least one object called a GL *context*. This object, which is an implicit parameter to every GL function, stores all of the GL related state for the application. Every texture, every buffer object, every enable, and much, much more is stored in the context. Since an application can have more than one context, the context to be used is selected by a window-system dependent function such as `glXMakeContextCurrent`.

In environments that implement OpenGL with X-Windows using GLX, every GL function, including the pointers returned by `glXGetProcAddress`, are *context independent*. This means that no matter what context is currently active, the same `glVertex3fv` function is used.

This creates the first bit of dispatch complexity. An application can have two GL contexts. One context is a direct rendering context where function calls are routed directly to a driver loaded within the application's address space. The other context is an indirect rendering context where function calls are converted to GLX protocol and sent to a server. The same `glVertex3fv` has to do the right thing depending on which context is current.

Highly optimized drivers or GLX protocol implementations may want to change the behavior of GL functions depending on current state. For example, `glFogCoordf` may operate differently depending on whether or not fog is enabled.

In multi-threaded environments, it is possible for each thread to have a different GL context current. This means that poor old `glVertex3fv` has to know which GL context is current in the thread where it is being called.

2. Overview of Mesa's Implementation

Mesa uses two per-thread pointers. The first pointer stores the address of the context current in the thread, and the second pointer stores the address of the *dispatch table* associated with that context. The dispatch table stores pointers to functions that actually implement specific GL functions. Each time a new context is made current in a thread, these pointers a updated.

The implementation of functions such as `glVertex3fv` becomes conceptually simple:

- Fetch the current dispatch table pointer.
- Fetch the pointer to the real `glVertex3fv` function from the table.
- Call the real function.

This can be implemented in just a few lines of C code. The file src/mesa/glapi/glapitemp.h contains code very similar to this.

```
void glVertex3f(GLfloat x, GLfloat y, GLfloat z)
{
    const struct _glapi_table * const dispatch = GET_DISPATCH();

    (*dispatch->Vertex3f)(x, y, z);
}
```

Sample dispatch function

The problem with this simple implementation is the large amount of overhead that it adds to every GL function call.

In a multithreaded environment, a naive implementation of GET_DISPATCH involves a call to pthread_getspecific or a similar function. Mesa provides a wrapper function called _glapi_get_dispatch that is used by default.

3. Optimizations

A number of optimizations have been made over the years to diminish the performance hit imposed by GL dispatch. This section describes these optimizations. The benefits of each optimization and the situations where each can or cannot be used are listed.

3.1. Dual dispatch table pointers

The vast majority of OpenGL applications use the API in a single threaded manner. That is, the application has only one thread that makes calls into the GL. In these cases, not only do the calls to pthread_getspecific hurt performance, but they are completely unnecessary! It is possible to detect this common case and avoid these calls.

Each time a new dispatch table is set, Mesa examines and records the ID of the executing thread. If the same thread ID is always seen, Mesa knows that the application is, from OpenGL's point of view, single threaded.

As long as an application is single threaded, Mesa stores a pointer to the dispatch table in a global variable called _glapi_Dispatch. The pointer is also stored in a per-thread location via pthread_setspecific. When Mesa detects that an application has become multithreaded, NULL is stored in _glapi_Dispatch.

Using this simple mechanism the dispatch functions can detect the multithreaded case by comparing _glapi_Dispatch to NULL. The resulting implementation of GET_DISPATCH is slightly more complex, but it avoids the expensive pthread_getspecific call in the common case.

```
#define GET_DISPATCH() \
    (_glapi_Dispatch != NULL) \
        ? _glapi_Dispatch : pthread_getspecific(&_glapi_Dispatch_key
)
```

Improved GET_DISPATCH Implementation

3.2. ELF TLS

Starting with the 2.4.20 Linux kernel, each thread is allocated an area of per-thread, global storage. Variables can be put in this area using some extensions to GCC. By storing the dispatch table pointer in this area, the expensive call to pthread_getspecific and the test of _glapi_Dispatch can be avoided.

The dispatch table pointer is stored in a new variable called _glapi_tls_Dispatch. A new variable name is used so that a single libGL can implement both interfaces. This allows the libGL to operate with direct rendering drivers that use either interface. Once the pointer is properly declared, GET_DISPACH becomes a simple variable reference.

```
extern __thread struct _glapi_table *_glapi_tls_Dispatch
    __attribute__((tls_model("initial-exec")));

#define GET_DISPATCH() _glapi_tls_Dispatch
```

TLS GET_DISPATCH Implementation

Use of this path is controlled by the preprocessor define GLX_USE_TLS. Any platform capable of using TLS should use this as the default dispatch method.

3.3. Assembly Language Dispatch Stubs

Many platforms has difficulty properly optimizing the tail-call in the dispatch stubs. Platforms like x86 that pass parameters on the stack seem to have even more difficulty optimizing these routines. All of the dispatch routines are very short, and it is trivial to create optimal assembly language versions. The amount of optimization provided by using assembly stubs varies from platform to platform and application to application. However, by using the assembly stubs, many platforms can use an additional space optimization (see *below*).

The biggest hurdle to creating assembly stubs is handling the various ways that the dispatch table pointer can be accessed. There are four different methods that can be used:

1. Using _glapi_Dispatch directly in builds for non-multithreaded environments.
2. Using _glapi_Dispatch and _glapi_get_dispatch in multithreaded environments.
3. Using _glapi_Dispatch and pthread_getspecific in multithreaded environments.
4. Using _glapi_tls_Dispatch directly in TLS enabled multithreaded environments.

People wishing to implement assembly stubs for new platforms should focus on #4 if the new platform supports TLS. Otherwise, implement #2 followed by #3. Environments that do not support multithreading are uncommon and not terribly relevant.

Selection of the dispatch table pointer access method is controlled by a few preprocessor defines.

- If GLX_USE_TLS is defined, method #3 is used.
- If HAVE_PTHREAD is defined, method #2 is used.
- If none of the preceding are defined, method #1 is used.

Two different techniques are used to handle the various different cases. On x86 and SPARC, a macro called GL_STUB is used. In the preamble of the assembly source file different implementations of the macro are selected based on the defined preprocessor variables. The assembly code then consists of a series of invocations of the macros such as:

```
GL_STUB(Color3fv, _gloffset_Color3fv)
```

SPARC Assembly Implementation of glColor3fv

The benefit of this technique is that changes to the calling pattern (i.e., addition of a new dispatch table pointer access method) require fewer changed lines in the assembly code.

However, this technique can only be used on platforms where the function implementation does not change based on the parameters passed to the function. For example, since x86 passes all parameters on the stack, no additional code is needed to save and restore function parameters around a call to pthread_getspecific. Since x86-64 passes

parameters in registers, varying amounts of code needs to be inserted around the call to `pthread_getspecific` to save and restore the GL function's parameters.

The other technique, used by platforms like x86-64 that cannot use the first technique, is to insert `#ifdef` within the assembly implementation of each function. This makes the assembly file considerably larger (e.g., 29,332 lines for `glapi_x86-64.S` versus 1,155 lines for `glapi_x86.S`) and causes simple changes to the function implementation to generate many lines of diffs. Since the assembly files are typically generated by scripts (see *below*), this isn't a significant problem.

Once a new assembly file is created, it must be inserted in the build system. There are two steps to this. The file must first be added to `src/mesa/sources`. That gets the file built and linked. The second step is to add the correct `#ifdef` magic to `src/mesa/glapi/glapi_dispatch.c` to prevent the C version of the dispatch functions from being built.

3.4. Fixed-Length Dispatch Stubs

To implement `glXGetProcAddress`, Mesa stores a table that associates function names with pointers to those functions. This table is stored in `src/mesa/glapi/glprocs.h`. For different reasons on different platforms, storing all of those pointers is inefficient. On most platforms, including all known platforms that support TLS, we can avoid this added overhead.

If the assembly stubs are all the same size, the pointer need not be stored for every function. The location of the function can instead be calculated by multiplying the size of the dispatch stub by the offset of the function in the table. This value is then added to the address of the first dispatch stub.

This path is activated by adding the correct `#ifdef` magic to `src/mesa/glapi/glapi.c` just before `glprocs.h` is included.

4. Automatic Generation of Dispatch Stubs

CHAPTER 37

Indices and tables

- genindex
- modindex
- search