

# Mali™ GPU Shader Development Studio

Version: 1.2.0

**User Guide**

**ARM®**

# Mali GPU Shader Development Studio

## User Guide

Copyright © 2009-2010 ARM. All rights reserved.

### Release Information

The following changes have been made to this book.

#### Change history

Date	Issue	Confidentiality	Change
14 October 2009	A	Non-Confidential	First release for v1.1
30 July 2010	B	Non-Confidential	First release for v1.2

### Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

### Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

### Product Status

The information in this document is final, that is for a developed product.

### Web Address

<http://www.arm.com>

# Contents

## Mali GPU Shader Development Studio User Guide

	<b>Preface</b>	
	About this book .....	ix
	Feedback .....	xi
<b>Chapter 1</b>	<b>Introduction</b>	
	1.1 About the Mali GPU Shader Development Studio .....	1-2
<b>Chapter 2</b>	<b>Installing the software</b>	
	2.1 Installation requirements .....	2-2
	2.2 Installation overview .....	2-3
	2.3 Download the installation package .....	2-4
	2.4 Configure the Eclipse Update Manager and install Shader Development Studio .....	2-6
	2.5 Configure the Shader Development Studio .....	2-10
	2.6 Uninstalling the Shader Development Studio Plug-in .....	2-16
<b>Chapter 3</b>	<b>The Shader Development Studio</b>	
	3.1 Opening the Shader Development Studio perspective .....	3-2
	3.2 Creating a new Shader effect .....	3-3
	3.3 Importing existing Shader effects and configurations .....	3-14
	3.4 Shader syntax checking .....	3-17
	3.5 Shader animation .....	3-19
<b>Chapter 4</b>	<b>Configuring the Shader Development Studio</b>	
	4.1 Configuration .....	4-2
	4.2 Wizards .....	4-6
	4.3 Editors .....	4-7
	4.4 Views .....	4-10

## Appendix A

### Shader Server

A.1	About the Shader Server .....	A-2
-----	-------------------------------	-----

### Glossary

# List of Tables

## Mali GPU Shader Development Studio User Guide

	Change history .....	ii
Table 4-1	Animation transport buttons .....	4-10
Table 4-2	Toolbar buttons .....	4-11
Table 4-3	Context buttons .....	4-11
Table 4-4	Camera control buttons .....	4-12
Table 4-5	Scene control buttons .....	4-12
Table 4-6	Toolbar control buttons .....	4-13
Table 4-7	Shader Attributes .....	4-13
Table 4-8	Toolbar control buttons .....	4-14

# List of Figures

## Mali GPU Shader Development Studio User Guide

Figure 2-1	Available Software dialog .....	2-6
Figure 2-2	Browse For Folder dialog .....	2-7
Figure 2-3	Select Mali Developer Tools in the Install dialog .....	2-8
Figure 2-4	Open Perspective dialog box .....	2-9
Figure 2-5	Shader Development Studio perspective loaded .....	2-9
Figure 2-6	Eclipse Preferences dialog box .....	2-10
Figure 2-7	Eclipse Preferences dialog box with Compiler pane .....	2-11
Figure 2-8	Default Renderers dialog box .....	2-12
Figure 2-9	Eclipse Preferences dialog box with Renderers pane .....	2-13
Figure 2-10	The Add Renderer dialog .....	2-13
Figure 2-11	The Add Renderer dialog with the emulator renderer .....	2-14
Figure 2-12	Eclipse Preferences dialog box with Linux renderer added .....	2-15
Figure 2-13	Plug-in removed .....	2-16
Figure 3-1	Shader Development Studio perspective .....	3-2
Figure 3-2	New Project dialog box .....	3-3
Figure 3-3	New project in Shader Development Studio .....	3-4
Figure 3-4	New Vertex Shader dialog box .....	3-5
Figure 3-5	Files generated by Eclipse Wizards .....	3-5
Figure 3-6	Shader Configuration Editor .....	3-6
Figure 3-7	Resources dialog box .....	3-7
Figure 3-8	Edited Shader Configuration Editor .....	3-7
Figure 3-9	Selected renderer in Shader Preview view .....	3-8
Figure 3-10	Selected shader in Shader Control view .....	3-8
Figure 3-11	Shader Attributes view .....	3-9
Figure 3-12	Matrix and Vector editor .....	3-9
Figure 3-13	Shader Uniforms view .....	3-10
Figure 3-14	Matrix and Vector editor .....	3-10
Figure 3-15	Render view of SimpleTest shader .....	3-11
Figure 3-16	simple.frag edit view .....	3-12
Figure 3-17	Shader Control view .....	3-13

Figure 3-18	Torus geometry .....	3-13
Figure 3-19	Existing source project .....	3-14
Figure 3-20	Example content in project .....	3-15
Figure 3-21	Demo - 04 - Lighting effect .....	3-16
Figure 3-22	Error indicator in shader source editor .....	3-17
Figure 3-23	Errors in Problems view .....	3-17
Figure 3-24	ARM geometry .....	3-19
Figure 3-25	Shader Control view .....	3-20
Figure 3-26	Shader Uniforms view .....	3-20
Figure 3-27	Environment Mapping animation .....	3-21
Figure 4-1	Shader Development Preferences pane .....	4-2
Figure 4-2	Compiler preferences pane .....	4-3
Figure 4-3	Renderers Preferences pane .....	4-4
Figure 4-4	Animation transport controls .....	4-10
Figure 4-5	Toolbar buttons .....	4-10
Figure 4-6	Performance Info dialog box .....	4-12

# Preface

This preface introduces the *Mali GPU Shader Development Studio User Guide*. It contains the following sections:

- *About this book* on page ix
- *Feedback* on page xi.

## About this book

This book is for the *Mali GPU Shader Development Studio User Guide*. This book is part of a suite belonging to the Mali Developer Tools.

## Intended audience

This book is written for first-time users of the Mali Shader Development Studio software. It assumes that the users are familiar with basic graphics processing concepts and the Eclipse software development environment.

## Using this book

This book is organized into the following chapters:

### Chapter 1 *Introduction*

Read this for an introduction to the Shader Development Studio software.

### Chapter 2 *Installing the software*

Read this for a description of how to install the Shader Development Studio software. The software is provided as a plug-in that can be installed into the Eclipse software development environment.

### Chapter 3 *The Shader Development Studio*

Read this for information on how to get started using the Shader Development Studio. This includes, setting up a new shader, importing an existing shader and using some of the features of the software.

### Chapter 4 *Configuring the Shader Development Studio*

Read this for a summary of all the functionality that is provided by the Shader Development Studio software.

### Appendix A *Shader Server*

Read this for information to use a remote renderer for the Shader Development Studio.

**Glossary** Read this for definitions of terms used in this book.

## Typographical conventions

The typographical conventions are:

<i>italic</i>	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
<b>bold</b>	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<u>monospace</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<i>monospace italic</i>	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
<b>monospace bold</b>	Denotes language keywords when used outside example code.

- < **and** >            Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example:
- MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode\_2>

## Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, <http://infocenter.arm.com>, for access to ARM documentation.

### ARM publications

This guide contains information that is specific to the Mali Developer Tools. See the following documents for other relevant information:

- *Mali GPU Developer Tools Technical Overview* (ARM DUI 501)
- *Mali GPU Performance Analysis Tool User Guide* (ARM DUI 0502)
- *Mali GPU Texture Compression Tool User Guide* (ARM DUI 0503)
- *Mali GPU User Interface Engine User Guide* (ARM DUI 0505)
- *OpenGL ES 1.1 Emulator User Guide* (ARM DUI 0506)
- *Mali GPU Binary Asset Exporter User Guide* (ARM DUI 0507)
- *Mali GPU Shader Library User Guide* (ARM DUI 0510)
- *OpenGL ES 2.0 Emulator User Guide* (ARM DUI 0511)
- *Mali GPU Offline Shader Compiler User Guide* (ARM DUI 0513).

### Other publications

This section lists relevant documents published by third parties:

- *OpenGL ES 1.1 Specification* at <http://www.khronos.org>.
- *OpenGL ES 2.0 Specification* at <http://www.khronos.org>.
- *OpenGL ES Shading Language Specification* at <http://www.khronos.org>.
- *OpenVG 1.1 Specification* at <http://www.khronos.org>.
- *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2* (5th Edition, 2005), Addison-Wesley Professional. ISBN 0-321-33573-2.
- *OpenGL Shading Language* (2nd Edition, 2006), Addison-Wesley Professional. ISBN 0-321-33489-2.

## Feedback

ARM welcomes feedback on this product and its documentation.

### Feedback on this product

If you have any comments or suggestions about this product, contact [malidevelopers@arm.com](mailto:malidevelopers@arm.com) and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

### Feedback on content

If you have comments on content then send an e-mail to [errata@arm.com](mailto:errata@arm.com). Give:

- the title
- the number, DUI0504B
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

# Chapter 1

## Introduction

This chapter provides information about the Mali GPU Shader Development Studio Tool, and describes how to install and start using the tool in your particular workflow. It contains the following section:

- *About the Mali GPU Shader Development Studio* on page 1-2.

## 1.1 About the Mali GPU Shader Development Studio

The Mali GPU Shader Development Studio is an Eclipse plug-in that extends the functionality of the Eclipse platform to enable editing of OpenGL ES 2.0 shaders. It can be used to develop shaders from scratch, or to work on existing shaders.

You can preview shaders as they are being developed by rendering them on remote OpenGL ES 2.0 hardware or on remote or local emulations.

The *OpenGL ES Shading Language* (ESSL) is used to develop shaders. For more information, see the *OpenGL ES Shading Language Specification*.

## Chapter 2

# Installing the software

This chapter describes how to install the Shader Development Studio. It contains the following sections:

- *Installation requirements* on page 2-2
- *Download the installation package* on page 2-4
- *Uninstalling the Shader Development Studio Plug-in* on page 2-16.

## 2.1 Installation requirements

This section lists the installation requirements for the Shader Developer Studio software. The sections included are:

- *JRE*
- *Eclipse*
- *Mali Developer Tools*.

---

**Note**

---

The Shader Developer Studio has been tested successfully on a 32-bit computer.

---

### 2.1.1 JRE

You must download and install the Java Runtime Environment (JRE) for either Windows XP or Linux from <http://www.java.com/>. The latest update of JRE6 is recommended.

### 2.1.2 Eclipse

The Shader Developer Studio software is an Eclipse plug-in.

#### **Eclipse**

Download and install Eclipse from <http://www.eclipse.org/>. Eclipse version 3.5 (at least) is recommended.

### 2.1.3 Mali Developer Tools

The Shader Development Studio software uses the Offline Shader Compiler, `malisc.exe`, to check the syntax of shader source files.

The Offline Shader Compiler is provided as a separate standalone tool. If the Offline Shader Compiler is not installed, the Shader Development Studio is not able to perform syntax checking of the shader sources or to gather performance statistics. You can download the Offline Compiler from the Mali Developer Center website identified in *Download the installation package* on page 2-4 and set its path in the configuration procedure, see *Set the location of the Offline Shader Compiler* on page 2-10.

The OpenGL ES 2.0 Emulator is required for local rendering on a workstation. See the *OpenGL ES 2.0 Emulator User Guide* for installation instructions.

For more information about the Mali Developer Tools, see the *Mali Developer Tools Overview*.

## 2.2 Installation overview

The Shader Development Studio plug-in is delivered as an Eclipse local update site package. To install the plug-in you can use the Eclipse update site mechanism. This is the recommended method for installing Eclipse plug-ins because it enables Eclipse to automatically manage the installation and un-installation of the plug-in, in addition to managing plug-in dependencies.

The installation procedure comprises of four steps:

1. Download the Shader Development Studio for either Windows or Linux versions.
2. Configure the Eclipse update manager with the location of the Shader Development Studio update site directory.

———— **Note** —————

The operating system used for this description was Windows, however with the exception to directory names, it is identical to using Linux. This is also applicable to steps 3 and 4.

3. Install the Shader Development Studio with Eclipse.

———— **Note** —————

Depending on your version of Eclipse:

- Select **Help** → **Software updates ...** from the Eclipse menu bar.
- Select **Help** → **Install New Software ...** from the Eclipse menu bar.

4. Configure the Shader Development Studio.

These steps are described in the following sections.

## 2.3 Download the installation package

Download the installation package for your operating system:

- *Windows installation*
- *Linux installation.*

### 2.3.1 Windows installation

Download the Microsoft Windows installation package:

1. Go to the Mali Developer Download Center website at:  
<http://www.malideveloper.com>
2. Download the following package:  
Mali\_GPU\_Shader\_Development\_Studio\_m.n.o.p\_Win32.msi  
where:  
**m** identifies the major version  
**n** identifies the minor version.  
**o.p** identifies the part and build version.
3. Select the package to download.
4. Run the file Mali\_GPU\_Shader\_Development\_Studio\_m.n.o.p\_Win32.msi
5. Select the required installation options and then click **Finish** to complete the installation.

By default, the Mali Shader Development Studio is installed in:

C:\Program Files\ARM\Mali Developer Tools\Mali GPU Shader Development Studio m.n.o

The Shader Development Studio update site directory is installed in:

C:\Program Files\ARM\Mali Developer Tools\Mali GPU Shader Development Studio m.n\updatesite

### 2.3.2 Linux installation

Download the Linux installation package:

1. Go to the Mali Developer Download Center website at:  
<http://www.malideveloper.com>
2. Download the following package:  
Mali\_GPU\_Shader\_Development\_Studio\_m.n.o.p\_Linux.tar.gz  
where:  
**m** identifies the major version  
**n** identifies the minor version.  
**o.p** identifies the part and build version.
3. To decompress the file:
  - open a command terminal and navigate to the directory where you have downloaded the package
  - type the following command:  
tar -zxvf Mali\_GPU\_Shader\_Development\_Studio\_m.n.o.p\_Linux.tar.gz

By default, the Shader Developer Studio is installed in:

ARM/Mali\_Developer\_Tools/Mali\_GPU\_Shader\_Development\_Studio\_m.n.o

The Shader Development Studio update site directory is installed in:

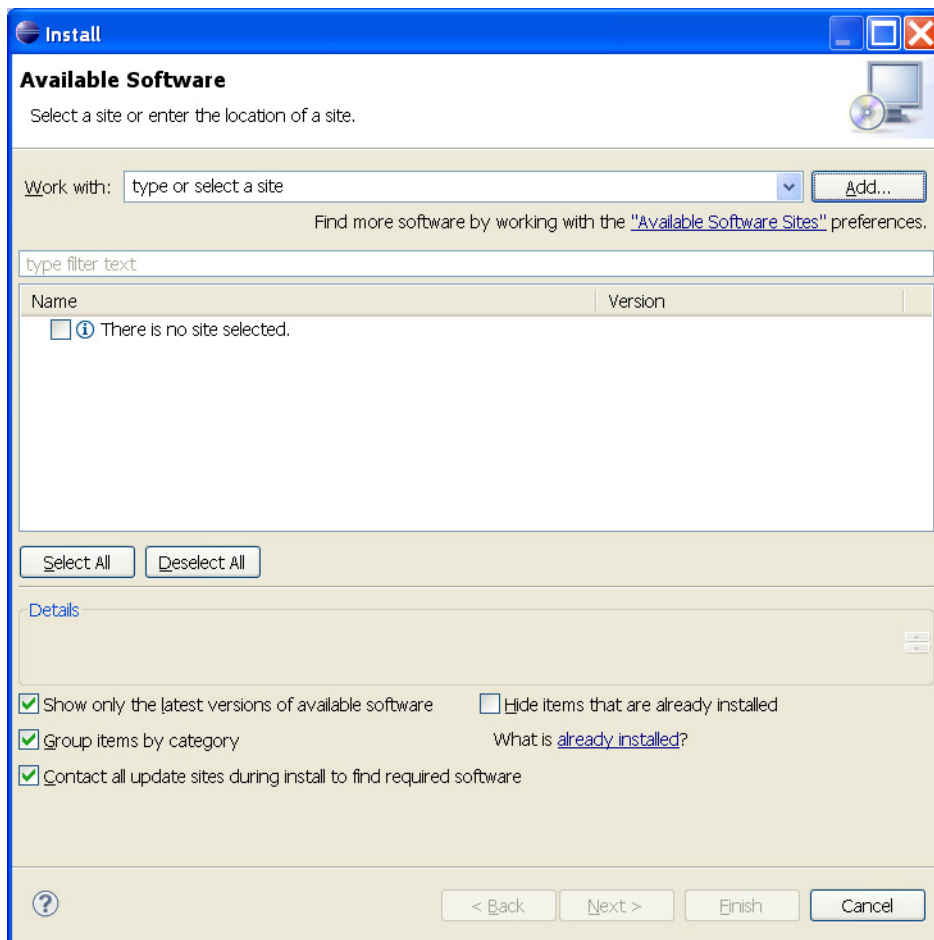
ARM/Mali\_Developer\_Tools/Mali\_GPU\_Shader\_Development\_Studio\_m.n.o/updatesite

## 2.4 Configure the Eclipse Update Manager and install Shader Development Studio

To configure the Eclipse Update Manager with the location of the update site directory for the Shader Development Studio plug-in:

1. Start Eclipse.
2. Select **Help** → **Install New Software ...** from the Eclipse menu bar.

The **Install Available Software** dialog box is displayed. See Figure 2-1:

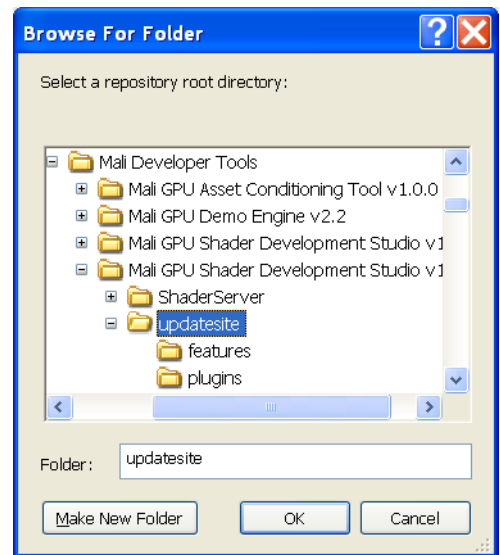


**Figure 2-1 Available Software dialog**

3. Click the **Add** button next to the **Work with** combo box.  
The Add Repository dialog is displayed.
4. Click **Local...** to display the Browse For Folder dialog.  
Browse to the location of the Shader Development Studio update site directory.
5. Navigate to the directory that contains the Shader Development Studio. See Figure 2-2 on page 2-7:

In this example, the Shader Development Studio update site directory is located in:

C:\Program Files\ARM\Mali Developer Tools\Mali GPU Shader Development Studio v1.2.0\updatesite



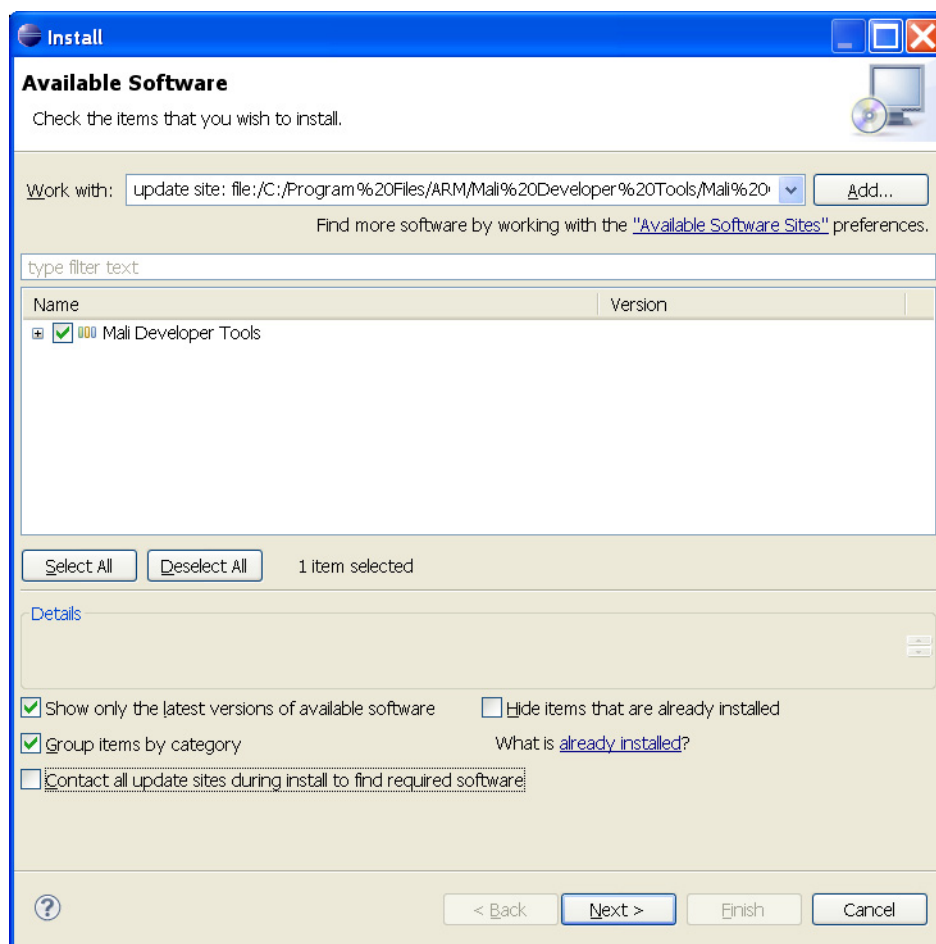
**Figure 2-2** Browse For Folder dialog

6. Click **OK**.

The **Add Repository** dialog box is updated with the location of the Shader Development Studio update site directory in the **Location** edit field.

7. Click **OK**.

The **Install Available Software** dialog box is updated with the location of the Shader Development Studio update site. See Figure 2-3 on page 2-8:



**Figure 2-3 Select Mali Developer Tools in the Install dialog**

8. Check the box next to **Mali Developer Tools**.
9. Click **Next** in the **Available Software** dialog box.  
The **Install** dialog box displays the item about to be installed.
10. Click **Finish** in the **Install Details** dialog box.  
A progress dialog box is displayed listing the components being installed  
The **Software Updates** dialog box is displayed requesting you to restart Eclipse to complete the installation.
11. Click **Yes** to restart Eclipse.  
If the installation is successful, the Shader Development Studio is loaded automatically when Eclipse restarts.
12. To confirm that the installation was successful, select **Window** → **Open Perspective** → **Other ...**.  
The **Open Perspective** dialog box is displayed. See Figure 2-4 on page 2-9:  
Check that **Shader Development Studio** is listed in the dialog box. If it is not listed, the Shader Development Studio is not installed correctly.



## 2.5 Configure the Shader Development Studio

The Shader Development Studio has many configuration options that are available from the Eclipse **Preferences** dialog box. See Chapter 4 *Configuring the Shader Development Studio* for more information.

To view the **Preferences** dialog box:

1. Select **Window** → **Preferences**.

The **Preferences** dialog box is displayed. The dialog box contains an entry for configuring the Shader Development Studio. See Figure 2-6:

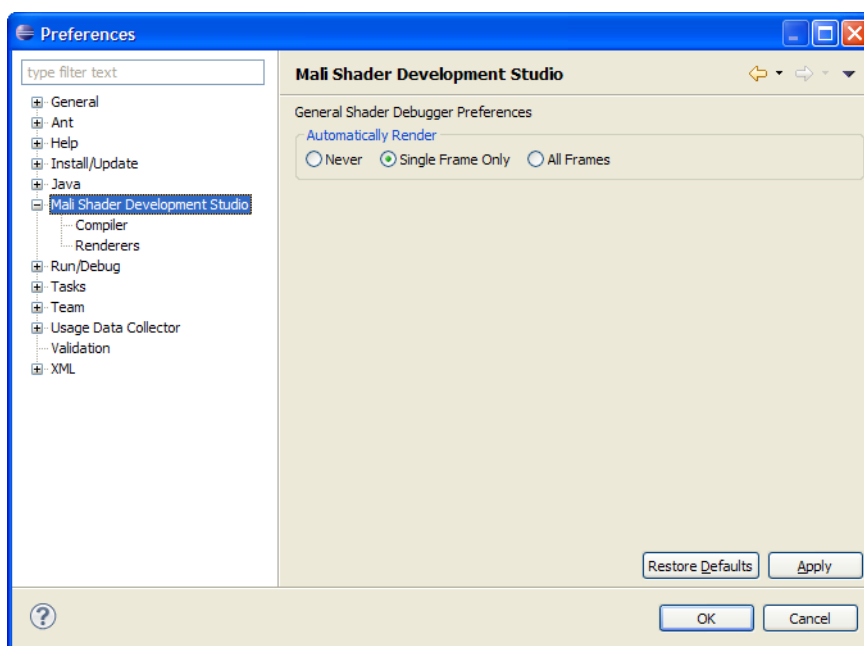


Figure 2-6 Eclipse Preferences dialog box

2. To configure the Shader Development Studio so that you can begin using it:
  - set the location of the Offline Shader Compiler
  - select a renderer to be used by the Shader Development Studio.

These configuration options are provided from the **Preferences** dialog box. See the following sections for more information.

### 2.5.1 Set the location of the Offline Shader Compiler

The Offline Shader Compiler is provided with the Mali Developer Tools. It enables the Shader Development Studio to check the syntax of shaders before they are sent for rendering. This works by compiling each shader in the background and gathering data on any warnings or errors that are generated.

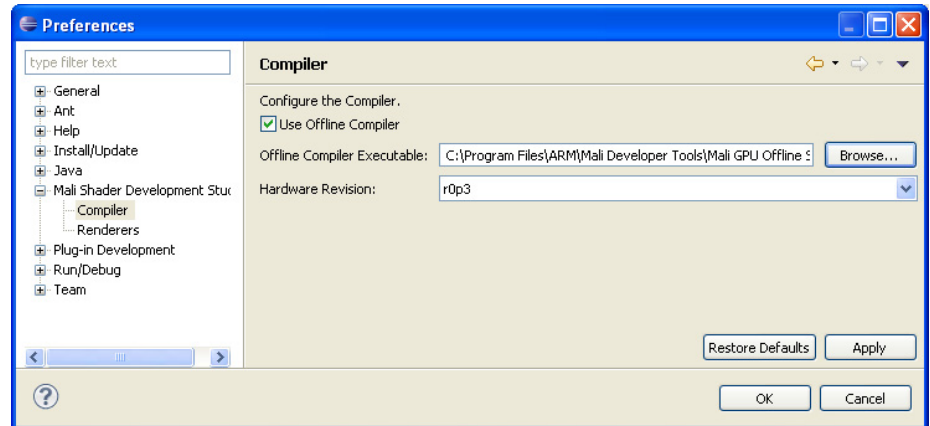
#### ————— Note —————

If you do not have the Mali Developer Tools installed on your computer, there is no offline compiler to configure, so you can skip this section. Be aware that because you do not have an Offline Shader Compiler installed, no syntax checking of the shaders is performed and no gathering of performance statistics is performed.

The Offline Shader Compiler is called `malisc.exe` and installed as part of the Mali Developer Tools. Set a path to where it is installed:

C:\Program Files\ARM\Mali Developer Tools\Mali GPU Offline Shader Compiler vm.n\bin

To set the correct configuration, click on **Shader Development Studio** → **Compiler** in the **Preferences** dialog box. See Figure 2-7:



**Figure 2-7 Eclipse Preferences dialog box with Compiler pane**

To provide the location of the offline compiler:

1. Select the **Compiler** preference pane from the **Preferences** dialog box.
2. Ensure the Offline Shader Compiler is enabled by checking the **Use Offline Compiler** box.
3. Click **Browse...** and browse to the new location of malisc.exe.
4. Click **OK** to set the new location.

## 2.5.2 Select a renderer

A renderer is used to perform the actual rendering of a shader effect. Any device that is capable of rendering OpenGL ES 2.0 shaders can be used for this purpose. This is a powerful concept that enables you to render your shader effects on local or remote OpenGL ES 2.0 emulations, or even on actual OpenGL ES 2.0 hardware, without performing any additional configuration.

The process depends on our operating system:

- *Select a renderer for Windows*
- *Select a renderer for Linux on page 2-13.*

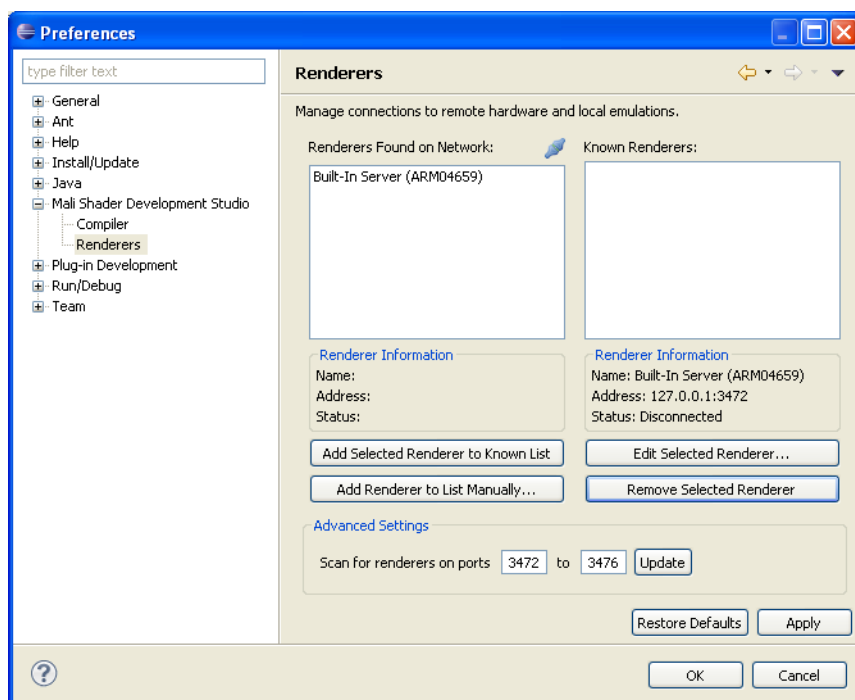
### Select a renderer for Windows

To select or add a renderer to be used by the Windows version of Shader Development Studio:

1. From the **Preferences** dialog box, select **Mali Shader Development Studio** → **Renderers** to display the **Renderers** preference pane.

Use the **Renderers** preference pane to manage connections to the renderers. In the Windows version, when this pane is open, it actively scans the local network for compatible devices.

In the Windows version, the Shader Development Studio is shipped with a local renderer, that is launched automatically at start-up by the Shader Development Studio. The local renderer appears in the **Renderers Found on Network** list as **Built-In Server**. See Figure 2-8 on page 2-12:



**Figure 2-8 Default Renderers dialog box**

**Note**

The local renderer requires that the OpenGL ES 2.0 Emulator version 1.2 is installed on your system.

2. Select the Built-In Server renderer.
3. Click **Add Selected Renderer to Known List**.

The Built-In Server is copied over to the **Known Renderers** list. See Figure 2-9 on page 2-13:

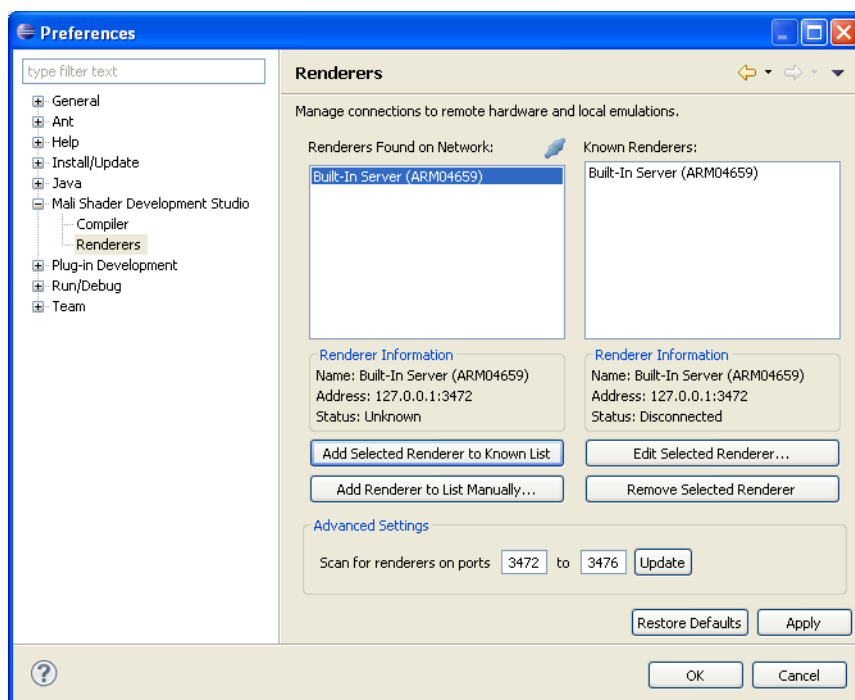


Figure 2-9 Eclipse Preferences dialog box with Renderers pane

If the device is not on the local network, you can add it manually by clicking **Add Renderer to List Manually....** See *Renderers* on page 4-3 for more information on how to add a new renderer.

4. Click **OK** to save the preferences that have been set and to close the **Preferences** dialog box.
5. The Shader Development Studio software is now configured.

### Select a renderer for Linux

To select or add a renderer to be used by the Linux version of Shader Development Studio:

1. From the **Preferences** dialog box, select **Mali Shader Development Studio** → **Renderers** to display the **Renderers** preference pane.
2. Click **Add Renderer to List Manually**. The Add Renderer dialog is displayed. See Figure 2-10:

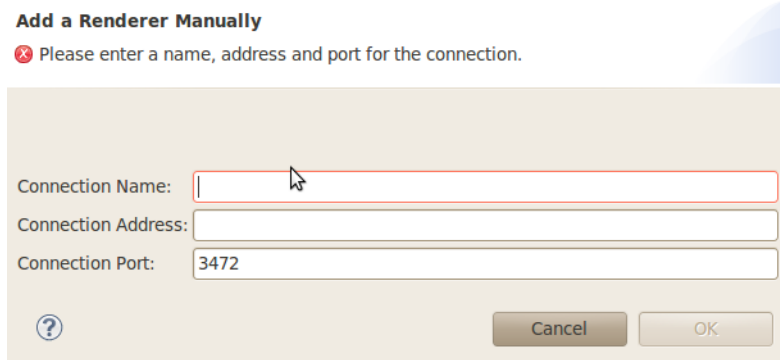


Figure 2-10 The Add Renderer dialog

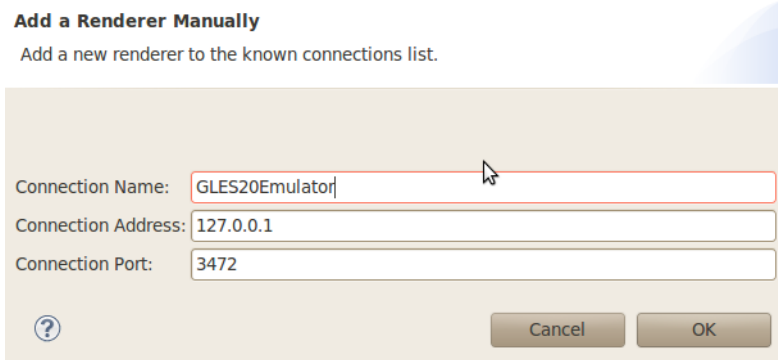
3. You can use either the renderer in the Emulator software or a renderer in a hardware device that contains a Mali GPU.

To use the Emulator renderer:

- a. Copy the Emulator binary to the local library directory:  

```
sudo cp <install_location>/ARM/Mali_Developer_Tools/OpenGL_ES_2_0_Emulator_1.2.0/bin/*.so /usr/local/lib
```
- b. Ensure the emulator libraries are picked up by the Shader Server:  

```
sudo /sbin/ldconfig
```
- c. Enter a name for the software renderer in the **Connection Name** field. See Figure 2-11.

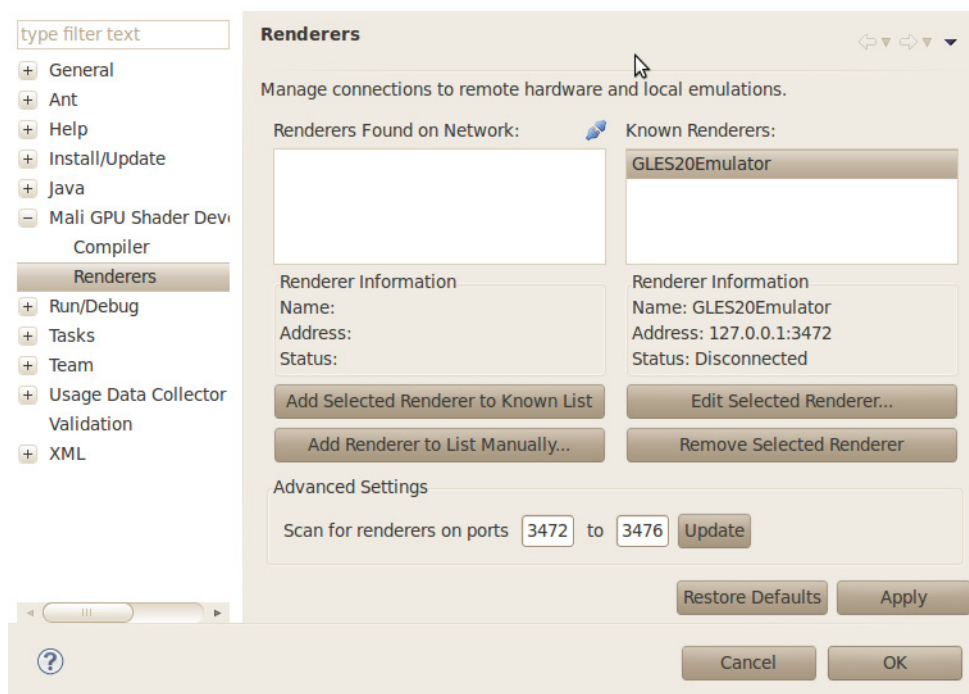


**Figure 2-11** The Add Renderer dialog with the emulator renderer

- d. Enter the local host address, which is always 127.0.0.1, in the **Connection Name** field.
- e. Enter a local port address in the Connection Port field. This address will be passed to the Shader Server when it starts.
- f. Click **OK**
- g. Restart eclipse if it is already open.

To use a hardware renderer in an attached system:

- a. Enter a name for the hardware renderer in the **Connection Name** field
  - b. Use `/sbin/ifconfig` on the target to get its internet address. Enter this value in the **Connection Address** field.
  - c. Enter the port address which you used to start the Shader Server on the target in the **Connection Port** field. See Appendix A *Shader Server*.
  - d. Click **OK**
4. The renderer is added to the **Known Renderers** list. See Figure 2-9 on page 2-13:



**Figure 2-12 Eclipse Preferences dialog box with Linux renderer added**

5. Click **OK** to save the preferences that have been set and to close the **Preferences** dialog box.
6. The Shader Development Studio software is now configured.

## 2.6 Uninstalling the Shader Development Studio Plug-in

To uninstall the Shader Development Studio plug-in for future updates:

1. Select **Help** → **Install new software** → **already installed**
2. Select **Mali Shader Development Studio** and click **Uninstall ...**
3. The uninstall details list is displayed. Select **Mali Shader Development Studio** and then click **Finish** to remove the plug-in.
4. Restart Eclipse and to confirm that the removal was successful, select **Window** → **Open Perspective** → **Other ...**. See Figure 2-13.

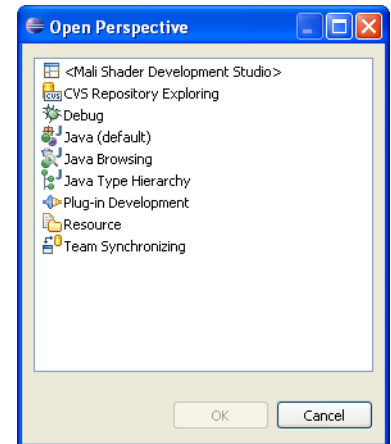


Figure 2-13 Plug-in removed

---

### Note

- With some installations you can see the icon <Mali Shader Development Studio> left in the open perspective window. If you click on this icon nothing happens, because the plug-in has already been correctly un-installed.
  - You can use the **Add or Remove** option in the Windows Control Panel to remove the local update site.
-

# Chapter 3

## The Shader Development Studio

This chapter describes how to get started using the Shader Developer Studio. It contains the following sections:

- *Opening the Shader Development Studio perspective* on page 3-2
- *Creating a new Shader effect* on page 3-3
- *Importing existing Shader effects and configurations* on page 3-14
- *Shader syntax checking* on page 3-17
- *Shader animation* on page 3-19.

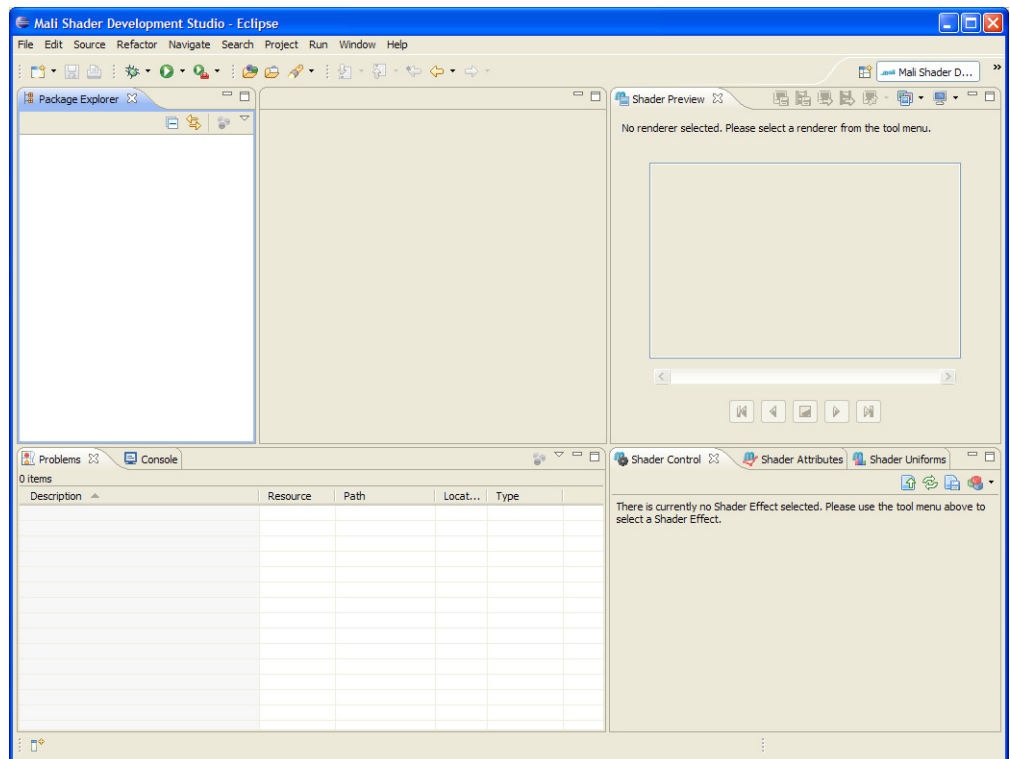
### 3.1 Opening the Shader Development Studio perspective

The Shader Development Studio software comes with a pre-built Eclipse perspective that can be used to start developing shaders for the Mali GPUs.

To open the Shader Development Studio perspective from Eclipse:

1. Start Eclipse.
2. Select **Window** → **Open Perspective** → **Other ...**.  
The **Open Perspective** dialog box is displayed.
3. Click **Shader Development Studio** in the **Open Perspective** dialog box.
4. Click **OK**.

The Shader Development Studio perspective is loaded. See Figure 3-1:



**Figure 3-1 Shader Development Studio perspective**

The views of the perspective can be moved and resized, docked and undocked, or dismissed altogether if they are not required.

To open individual views, select **Window** → **Show View** and select the view you require. For more information about the views provided by the Shader Development Studio, see *Views* on page 4-10.

## 3.2 Creating a new Shader effect

This section describes how to use the Shader Development Studio to create a new shader effect.

A shader effect is a collection of vertex and fragment shaders that make up a single shader program that can be rendered on a target. A project can have multiple shader effects, each of these can share shader source files. The shader effects are contained in and managed by a Shader Configuration File. A project can only have one shader configuration file, that must exist at the root level of the project.

### 3.2.1 Creating a new Vertex and Fragment Shader

To create a new shader effect, you must create the source files that comprise the effect and then add the source files to a project.

The Shader Development Studio works with any kind of project that can be created in Eclipse including a Java Project and a C Project. Eclipse provides wizards for creating these projects, and the Shader Development Studio provides wizards to guide you through the creation of shader-specific resources. For more information about the wizards provided by the Shader Development Studio, see *Wizards* on page 4-6.

For example, to create a new Java project with a Vertex Shader and Fragment Shader:

1. Select **File** → **New** → **Project ...**

The **New Project** dialog box is displayed. See Figure 3-2:

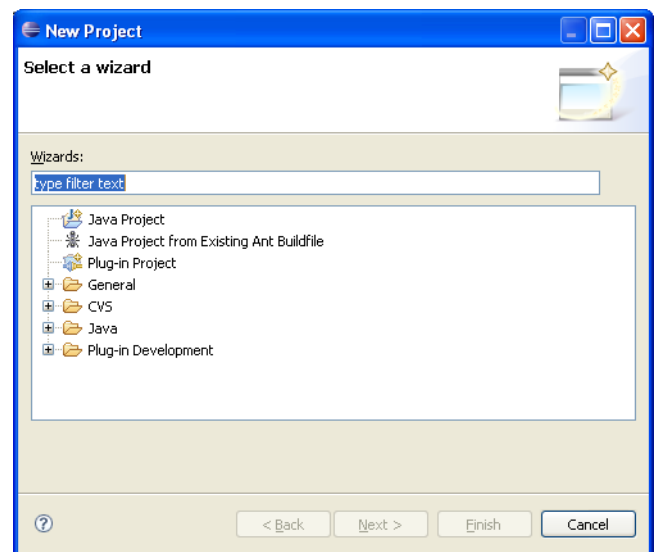


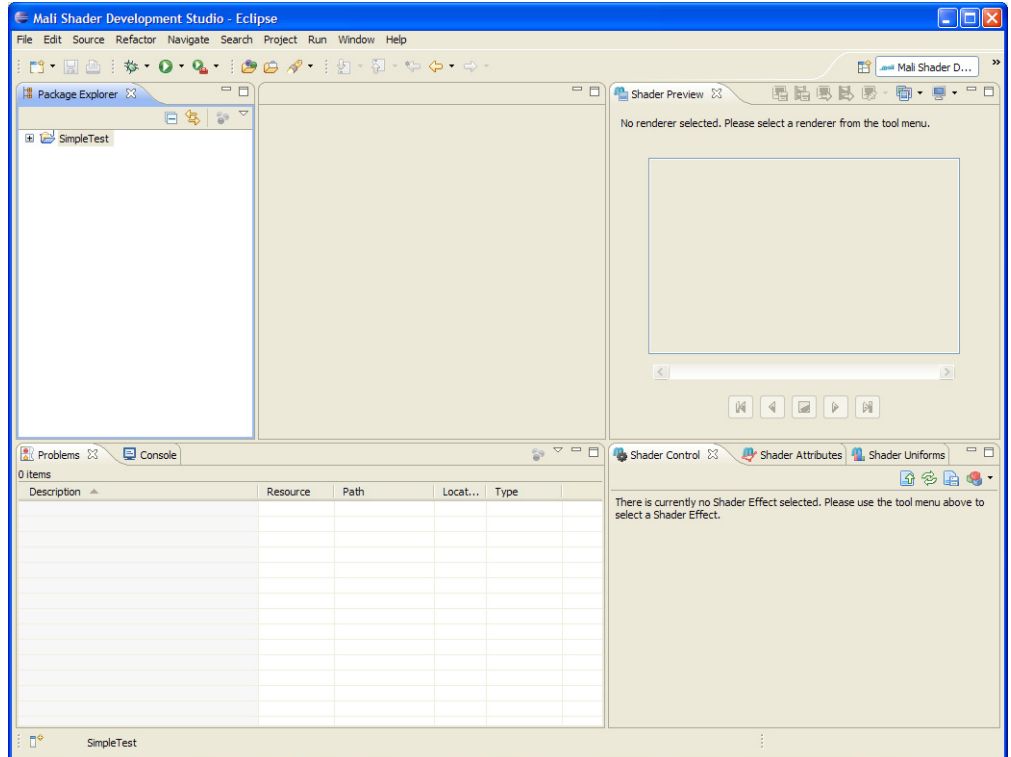
Figure 3-2 New Project dialog box

2. Click **Java Project**.
3. Click **Next**.  
The **New Java Project - Create a Java project** dialog box is displayed.
4. Enter a name for the project. For example, **SimpleTest**.
5. Click **Next**.  
The **New Java Project - Java Settings** dialog box is displayed.
6. Click **Finish**.

The **Open Associated Perspective** dialog box is displayed asking if you want to open the Java perspective.

7. Click **No**.

The new project is created in the Package Explorer pane of the Shader Development Studio. See Figure 3-3:



**Figure 3-3** New project in Shader Development Studio

8. Click the + icon next to the **SimpleTest** project in the Package Explorer pane to expand the contents.
9. Right-click on the **src** directory and select **New** → **Vertex Shader**.  
The Vertex Shader dialog box is displayed. See Figure 3-4 on page 3-5:

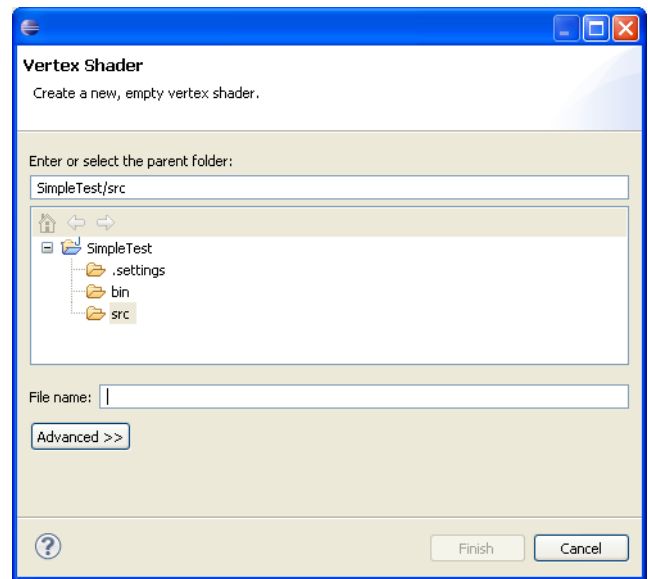


Figure 3-4 New Vertex Shader dialog box

10. Enter a name for the shader, for example **simple.vert**, in the **Filename** edit field
11. Click **Finish**.  
The vertex shader is created in the **src** directory.
12. Right-click on the **src** directory again and select **New** → **Fragment Shader**. The Fragment Shader dialog box is displayed.
13. Enter a name for the shader, for example **simple.frag**, in the **Filename** edit field.
14. Click **Finish**.  
The fragment shader is created in the **src** directory.
15. Two files are created in the **src** directory that contain the source code for a trivial shader effect. See Figure 3-5:

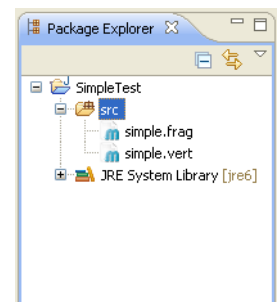


Figure 3-5 Files generated by Eclipse Wizards

### 3.2.2 Creating a new Shader Configuration File

This section describes how to create a shader configuration file. The configuration file is required to link the fragment shader and vertex shader source files together to produce an overall effect. The shader configuration file configures the type of effect that is produced.

To create a new shader configuration file:

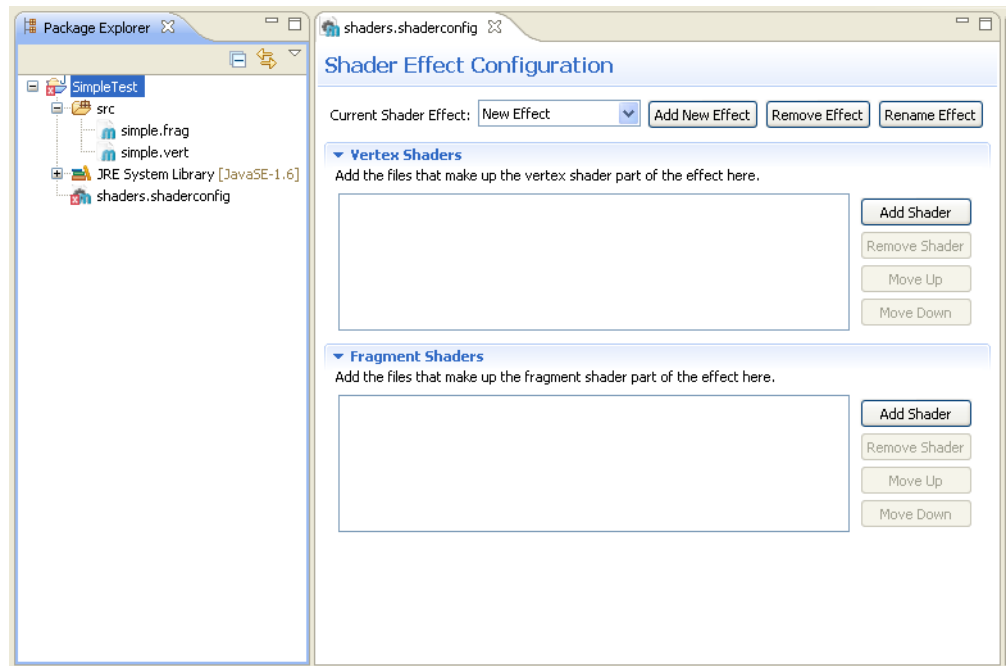
1. Right-click on the **SimpleTest** project in the **Package Explorer** pane and select **New → Shader Configuration**.

The **Shader Configuration** dialog box is displayed.

2. Click **SimpleTest**.

3. Click **Finish**.

The Shader Configuration file (named `shaders.shaderconfig`) is created in the **SimpleTest** project and the **Shader Configuration Editor** pane is displayed. See Figure 3-6:



**Figure 3-6 Shader Configuration Editor**

For more information about the shader configuration editor, see *Shader Configuration editor* on page 4-7.

The shader configuration editor enables you to manage shader effects for a project. A default effect, called **New Effect**, is created when the shader configuration is created. The default effect is used for this example.

The shader configuration editor separates vertex and fragment shaders into two lists. You can add any source file in the project to these lists. This enables *utility* shader sources, that are neither specifically vertex or fragment shader sources, to be reused.

4. Click **Add Shader** in the **Vertex Shaders** section.  
The **Resource Selection** dialog box is displayed.
5. Click the + icon next to **SimpleTest** and then click the **src** folder. See Figure Figure 3-7 on page 3-7:

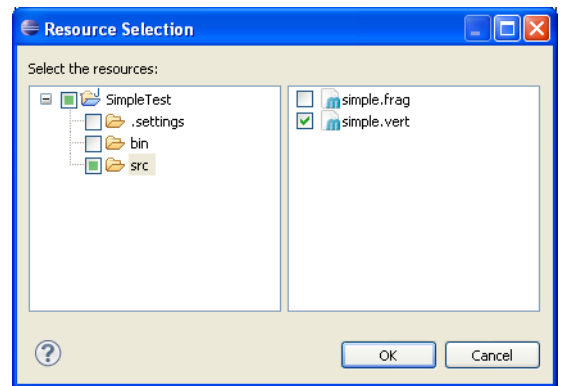


Figure 3-7 Resources dialog box

6. Click the check box next to **simple.vert**.
7. Click **OK**.  
The vertex shader source is added to the effect.
8. To add a shader to the Fragment shader section:
  - a. Click **Add Shader** in the **Fragment Shaders** section.
  - b. Click the + icon next to **SimpleTest** and then click the **src** folder.
  - c. Click **simple.frag** in the **Resource Selection** dialog box.
  - d. Click **OK**.
9. The **Shader Configuration Editor** contains the shaders. See Figure 3-8:

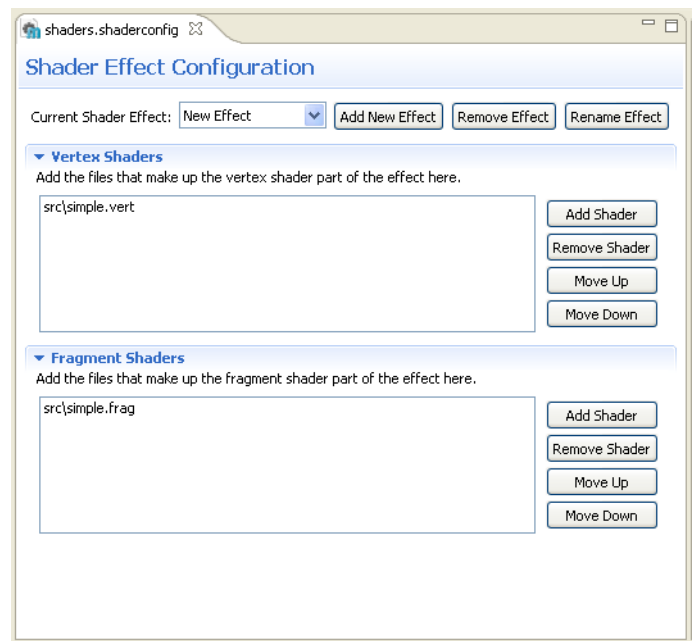


Figure 3-8 Edited Shader Configuration Editor

10. Select **File** → **Save** to save the configuration.  
You have created your first shader effect called **New Effect**.


———— **Note** ————

Eclipse displays warning messages in the **Problems** pane when the shader effect is created. These warnings are a result of uninitiated uniform and attribute variables in the shader programs. These can be ignored and eventually disappear after completing the *Rendering the shader effect* section.

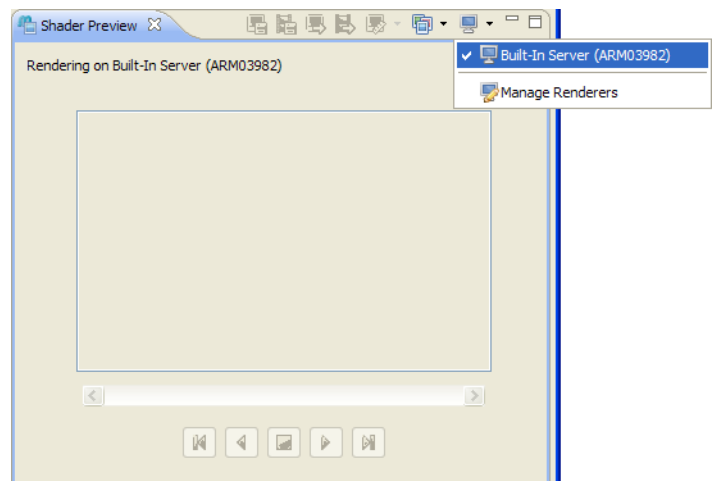
### 3.2.3 Rendering the shader effect

The shader effect created in the previous section can be rendered in the Shader Preview view. This view shows the result of rendering a shader on the selected target.


To render the shader effect:

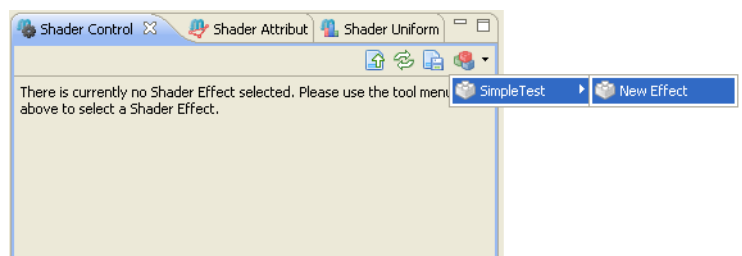
1. If it is not already open, open the Shader Preview view. To open this view, select **Window** → **Show View** and select the **Shader Preview** view.
2. In the **Shader Preview** view, click the Select Renderer icon  and select an appropriate renderer from the menu displayed.

A tick appears next to the renderer that is currently in use. See Figure 3-9:



**Figure 3-9 Selected renderer in Shader Preview view**

3. In the Shader Control view, click the Select Shader to Render icon  and select **SimpleTest** → **New Effect**. See Figure 3-10:



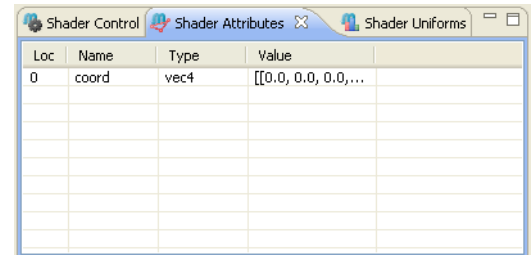
**Figure 3-10 Selected shader in Shader Control view**

A Progress bar is displayed in the Shader Preview view to indicate that the selected shader is being rendered by the target. The rendered image is then displayed in the Shader Preview view.

———— **Note** ————

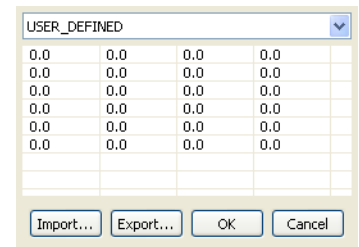
Although the image is rendered, no actual image is displayed in the view. This is because the geometry vertices and the camera matrix have not yet been set for the selected shader. To fix this, adjust the values of the uniforms and attributes in the shader. Two warning messages are issued about this in the **Problems** view, if you have it open. To assign values to attributes and uniforms use the Shader Attributes and Shader Uniforms views.

4. Click the **Shader Attributes** view alongside the **Shader Control** view.  
The Shader Attributes view is displayed. See Figure 3-11:



**Figure 3-11 Shader Attributes view**

5. In the **Value** column, click the cell corresponding to the value of the coord attribute.  
A button is displayed in the cell.
6. Click the button.  
The Matrix and Vector editor is displayed. See Figure 3-12:



**Figure 3-12 Matrix and Vector editor**

7. Select **VERTICES** from the drop-down menu.
8. Click **OK**.  
This action assigns vertex values for the currently selected geometry to the coord attribute.

———— **Note** ————

The warning message about this is automatically removed from the Problems view.

9. Click the Shader Uniforms tab alongside the Shader Attributes tab.  
The Shader Uniforms tab is displayed. See Figure 3-13 on page 3-10:

Loc	Name	Type	Initial Value	Final Value
0	camera	mat4	[[1.0, 0.0, 0.0, ...	[[1.0, 0.0, 0.0, ...

Figure 3-13 Shader Uniforms view

- In the **Initial Value** column, click the cell corresponding to the value of the camera uniform required for inspection and modification.

A button is displayed in the cell.

- Click the button.

The Matrix and Vector editor is displayed. See Figure 3-14:

USER_DEFINED			
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Import... Export... OK Cancel

Figure 3-14 Matrix and Vector editor


- Select **CAMERA\_MATRIX** from the drop-down menu.

- Click **OK**.

This action assigns values to the camera uniform.

———— **Note** —————

The warning message about this is automatically removed from the Problems view.

- Click the **Render Single Frame** icon  in the Shader Preview view.

A graduated-color square is displayed in the preview. See Figure 3-15 on page 3-11:

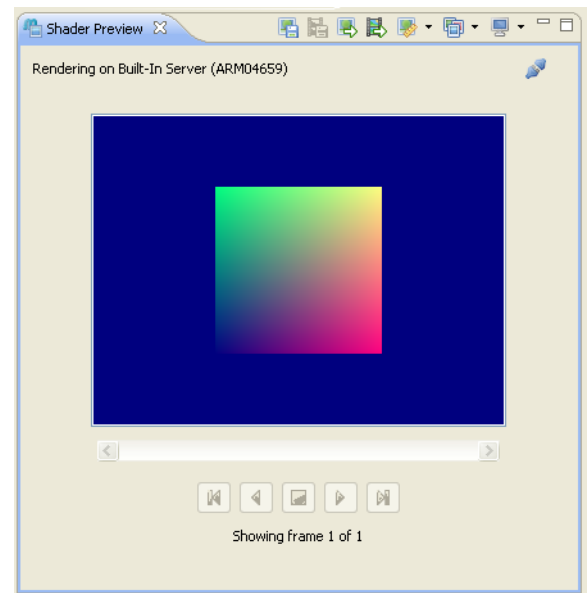



Figure 3-15 Render view of SimpleTest shader

### 3.2.4 Editing the effect

This section describes how to edit the shader effect that was produced in the previous section.

To edit the shader effect:

1. In the **Shader Preview** view, click the Default Rendering Mode icon  and select **Automatically Render One Frame** from the menu displayed.

This setting causes the **Shader Preview** view to update the preview when the components of the shader effect are changed. For example:

- shader source files
  - shader attributes
  - shader uniforms.
2. In the **Package Explorer** view, double-click `simple.frag` in the `src` directory. The `simple.frag` shader source file is opened in Eclipse.

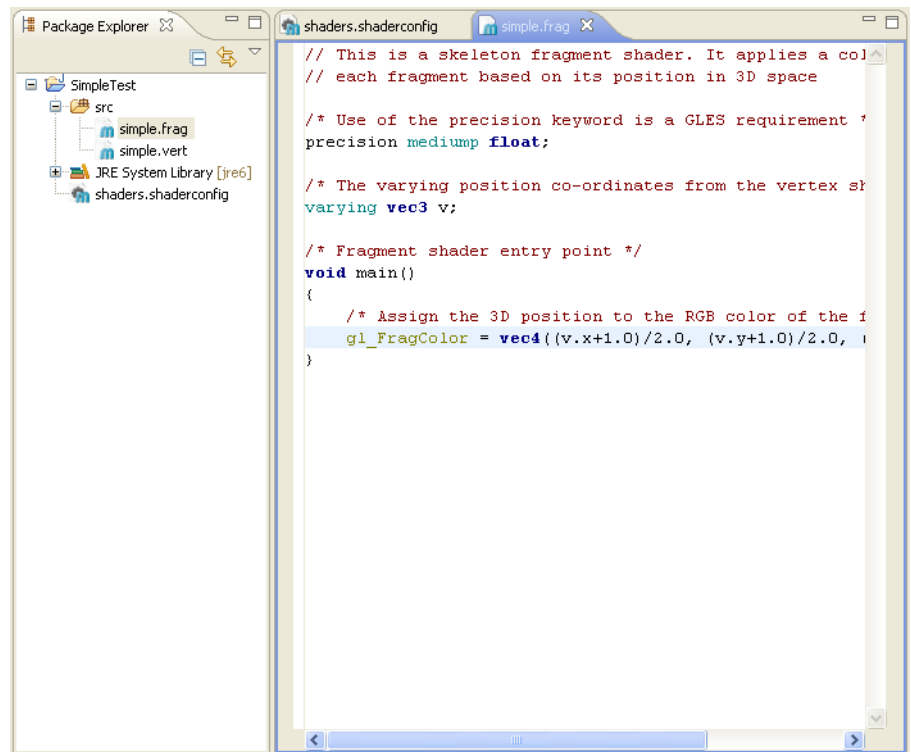


Figure 3-16 simple.frag edit view

3. Change the line:  
`gl_FragColor = vec4((v.x+1.0)/2.0, (v.y+1.0)/2.0, (v.z+1.0)/2.0, 1.0);`  
to:  
`gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);`
4. Select **File** → **Save** to save the file,  
The **Shader Preview** view is updated to show a solid white square instead of the graduated-color square.
5. Change the edited line back to the original version. You can use **Ctrl-Z** to undo the changes made.  
The **Shader Preview** view is updated to show the original graduated color square.

### 3.2.5 Rendering with different geometries

Currently, the shader effect is being rendered onto a simple geometric plane. The Shader Development Studio comes with several pre-built models that you can use to render to instead of a simple plane. The models include basic shapes such as a sphere and cube, to more complex shapes like a rock and a chess piece.

To render using a different geometry, such as a torus:

1. Click on the **Shader Control** view.  
The **Shader Control** view is displayed. See Figure 3-17 on page 3-13:

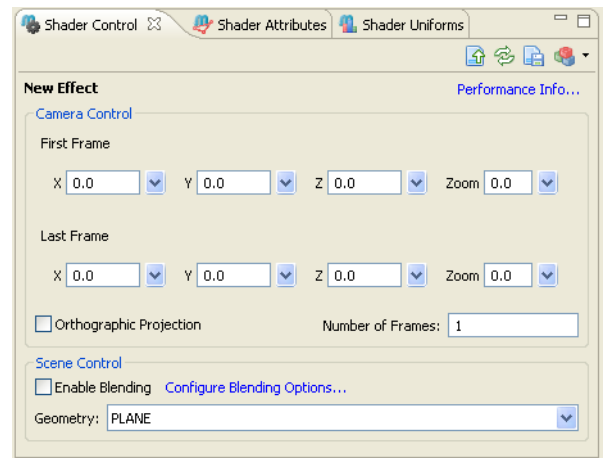


Figure 3-17 Shader Control view

**Note**

If necessary, resize the **Shader Control** view to display the **Scene Control** group box.

2. Select TORUS from the **Geometry** drop-down list box in the **Scene Control** box. The **Shader Preview** window is updated to show a torus with the shader effect rendered on its surface. See Figure 3-18:

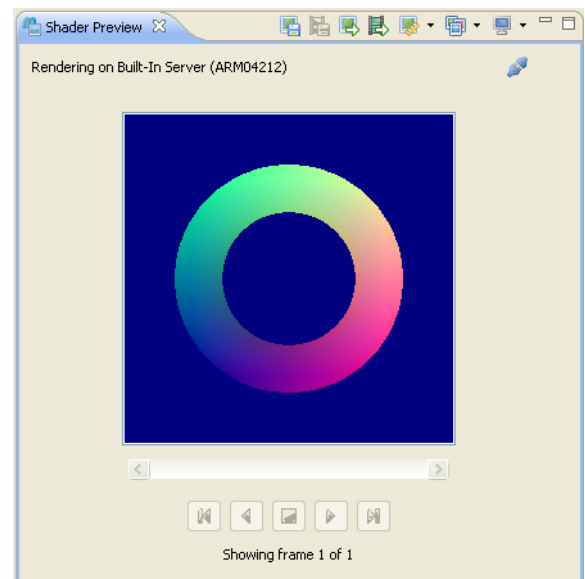


Figure 3-18 Torus geometry

The torus is a three-dimensional geometry, but it looks two-dimensional, see Figure 3-18. This is because the simple shader effect does not perform any lighting calculations, so no shadows are present to give depth cues.

In the next section, an advanced shader example is used that implements lighting calculations and shows the torus is a three-dimensional geometry.

### 3.3 Importing existing Shader effects and configurations

To create a new Generic Project in Eclipse and import the example shader content from the Mali Shader Library into this project:

1. Select **File** → **New** → **Project ...**  
The **New Project** dialog box is displayed.

2. Click **Java Project**.

3. Click **Next**.

The **New Java Project - Create a Java project** dialog box is displayed.

4. Enter a name for the project. For example, **ExampleContent**.

———— **Note** ————

The project is set to **Create a new project from workspace**. To avoid a known issue documented in the *Mali GPU Shader Development Studio Errata*, ARM recommends you choose the option **Create project from existing source** while creating new eclipse java project for the shader sources. Only the **Create project from existing source** is described.

5. Copy the ShaderExamples folder to your work area, it is located at:  
C:\Program Files\ARM\Mali Developer Tools\Mali GPU Shader Library vm.n

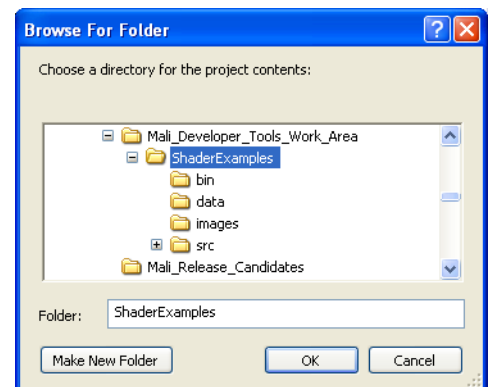
In this example the location for the work area is:

C:\Mali Developer Tools Work Area\

———— **Note** ————

This is done to prevent any data corruption occurring in the original folder.

6. Select **Browse** and locate your directory. In this example:  
C:\Mali Developer Tools Work Area\ShaderExamples. See Figure 3-19:



**Figure 3-19 Existing source project**

7. Select **OK**.

8. Click **Next**.

The **New Java Project - Java Settings** dialog box is displayed.

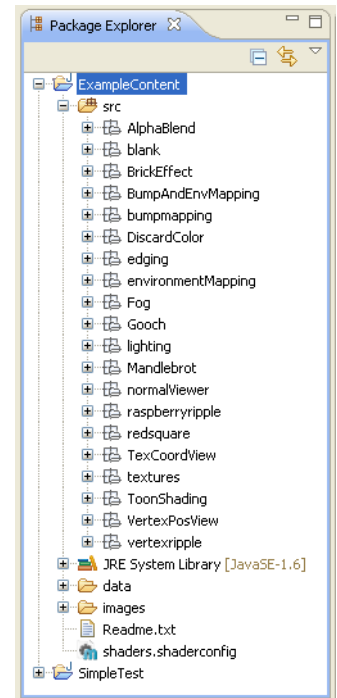
9. Click **Finish**.

The **Open Associated Perspective** dialog box is displayed asking if you want to open the Java perspective.

10. Click **No**.

The new project is created in the Package Explorer pane of the Shader Development Studio.

11. The selected files are imported into the project. See Figure 3-20:




**Figure 3-20 Example content in project**

———— **Note** ————

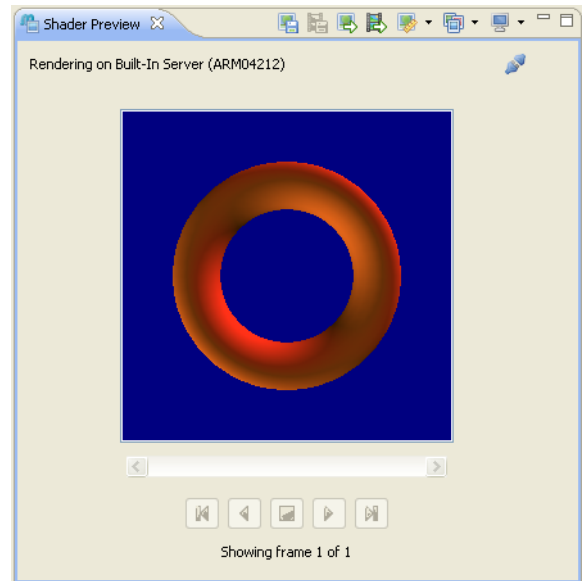
The steps to create a new Eclipse project for OrangeBookExamples, also present in Mali Shader, are similar.

The example content available in the Shader Library contains some more advanced shaders that implement the complex graphics effects that were missing from the example shader effect used in previous sections.

A pre-built shader configuration file is supplied with the example content that contains pre-made shader effects, so you do not have to create a shader configuration manually. Instead, the shader effects can be rendered immediately.

12. In the **Shader Control** view, click the Select Shader to Render icon  and select **ExampleContent** → **Demo 04 - Lighting**.

The torus is re-rendered in the **Shader Preview** view, but this time lighting calculations are applied. The torus is now three-dimensional. See Figure 3-21 on page 3-16:



**Figure 3-21 Demo - 04 - Lighting effect**

Try changing the preview model in the **Shader Control** view to see the effects that are produced by the other geometry models provided in the example content.

### 3.4 Shader syntax checking

If you do not have the Offline Compiler available, you can skip this section. For more information about the Offline Compiler, see *Mali GPU Developer Tools Overview*.

Each time a shader source file is saved, the Shader Development Studio compiles the shader in the background using the offline compiler. If any errors are found, error messages are displayed in:

- Problems view.
- Alongside the offending line of the shader source file, if it is open in the shader source editor.

To see how errors are displayed by Eclipse:

1. Open the file `simple.frag` in the SimpleTest project.
2. Remove the semi-colon at the end of the `vec4` expression.
3. Save the file.

A red error marker is displayed on the line beneath the `vec4` expression line. See Figure 3-22:

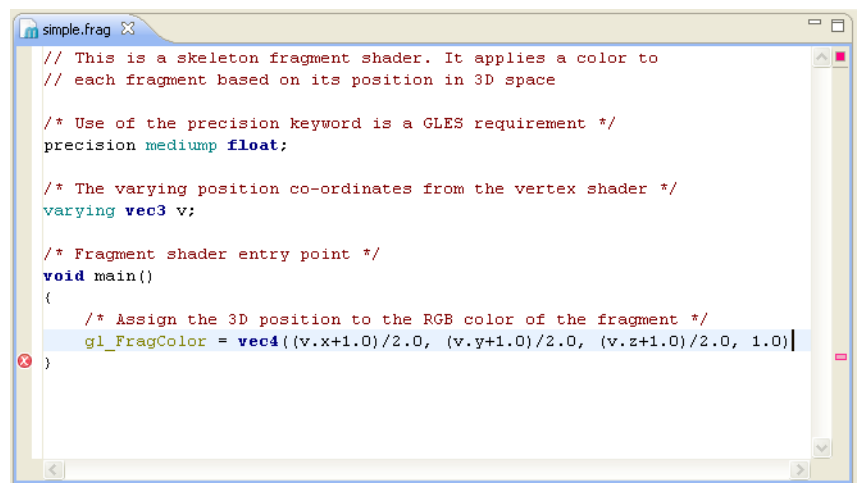


Figure 3-22 Error indicator in shader source editor

Similar to a C compiler, the offline compiler reports a missing semicolon as an error on the next line.

The removal of the semi-colon generates four separate error messages in the Problems view. See Figure 3-23:

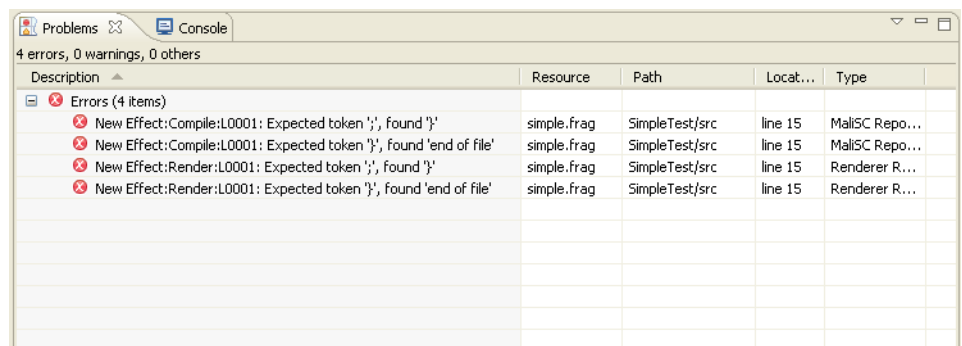


Figure 3-23 Errors in Problems view

4. Replace the semi-colon and save the file.

The source file is compiled in the background by the offline compiler, because there are no errors in the source code, the error marker is removed from the source editor and the error messages are removed from the **Problems** view.

## 3.5 Shader animation

The Shader Development Studio also supports shader animation. This can be used to see how a shader effect changes over time. This section describes how to set up a shader for animation and how to animate the shader.

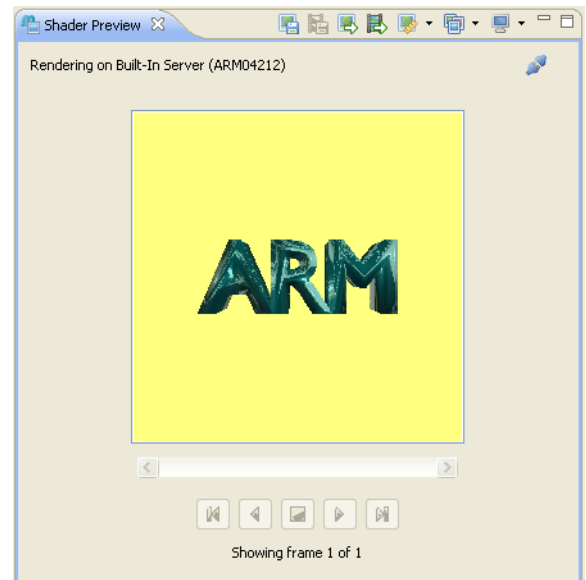
### 3.5.1 Setting up a shader to use animation

A shader is animated using a key framing method. Values are set for the initial frame and final frame, the Shader Development Studio then interpolates the values between these two frames to create the animation effect.

The *Environment Mapping* shader demo provided in the content directory is set up to use animation. To view how animation has been setup for this shader:

1. Open the project that you created containing the Example content provided with the Shader Development Studio software. See *Importing existing Shader effects and configurations* on page 3-14.
2. In the **Shader Control** view, select **ExampleContent** → **Demo 07 - Environment Mapping**.

The Shader Preview is updated to show a custom geometry called *ARM*, with environment mapping applied. See Figure 3-24:



**Figure 3-24 ARM geometry**

3. The **Shader Control** view and the Shader Uniforms view set up the key framing parameters for the shader animation.

In the **Camera control** group box in the Shader Control view, values are set for:

- camera position for the first frame, this is also the camera position used to render single-frame shaders
- camera position of the last frame and the number of frames to render. See Figure 3-25 on page 3-20:

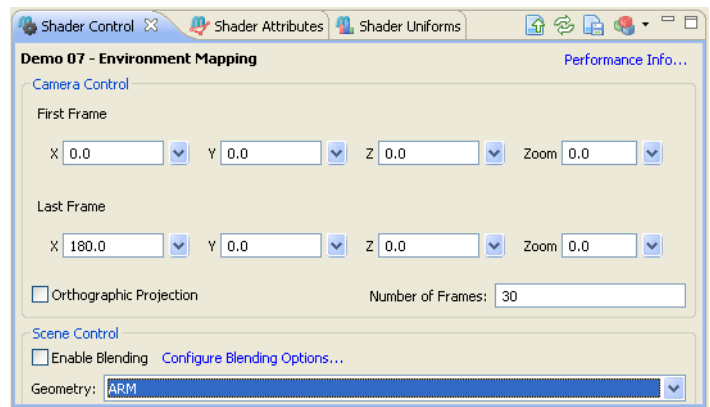


Figure 3-25 Shader Control view

In the **Shader Uniforms** view, initial values and final values are set for each uniform listed. These are the values that are interpolated to make the animation. See Figure 3-26:


Loc	Name	Type	Initial Value	Final Value
2	lightPosition	vec3	[1.0, 1.0, 1.0]	[1.0, 1.0, 1.0]
4	modelview	mat4	MODELVIEW_M...	
1	envMap	sampler2D	atrium.png	
7	surfaceC...	vec3	[0.0, 1.0, 1.0]	[0.0, 1.0, 1.0]
6	proj	mat4	PROJECTION_...	
3	mixRatio	float	0.3	0.3
0	ambient	float	0.3	0.3
5	normalMa...	mat3	NORMAL_MATRIX	

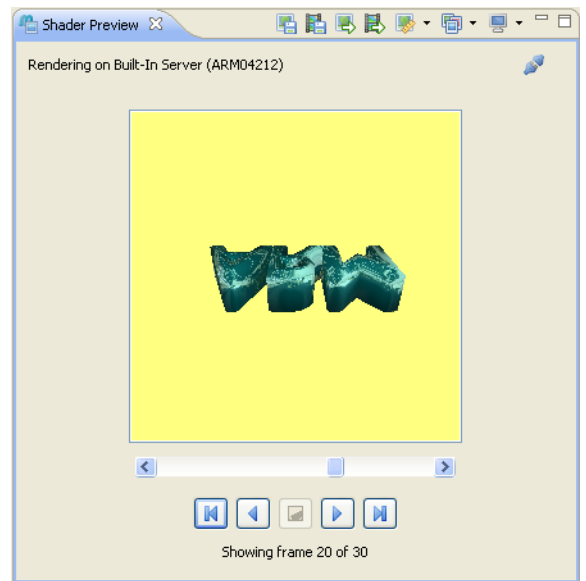
Figure 3-26 Shader Uniforms view

### 3.5.2 Animating the shader

This section describes how to animate the *Environment Mapping* shader demo.

The *Environment Mapping* shader demo effect already has key frame values set, so the animation can be generated immediately. To render all frames:

1. Click the *Render All Frames* icon  in the **Shader Preview** view.  
A Progress bar is displayed in the **Shader Preview** view to indicate that the selected shader is being rendered by the target. The first frame of the animation is displayed in the **Shader Preview** view.
2. Use the transport controls beneath the preview to control the animation.  
Click *Play Forward* to start the animation. The shader preview shows the number of frames requested, interpolated between the initial frame and final frame. See Figure 3-27 on page 3-21:



**Figure 3-27 Environment Mapping animation**

For more information about animating shaders, see *Shader Control* on page 4-11 and *Shader Uniforms* on page 4-13.

# Chapter 4

## Configuring the Shader Development Studio

This chapter describes all the options that are available for configuring the behavior of the Shader Development Studio. The sections in this chapter are:

- *Configuration* on page 4-2
- *Wizards* on page 4-6
- *Editors* on page 4-7
- *Views* on page 4-10.

## 4.1 Configuration

This section describes the general configuration options for the Shader Development Studio.

To view all the options available, select **Preferences** from the **Window** menu. The **Preferences** dialog box is displayed.

The sections describe the options displayed in this dialog box.

### 4.1.1 Shader Development Studio

To view the options for the Shader Development Studio, click **Shader Development Studio** in the **Preferences** dialog box.

This pane sets general options for the Shader Development Studio. See Figure 4-1:

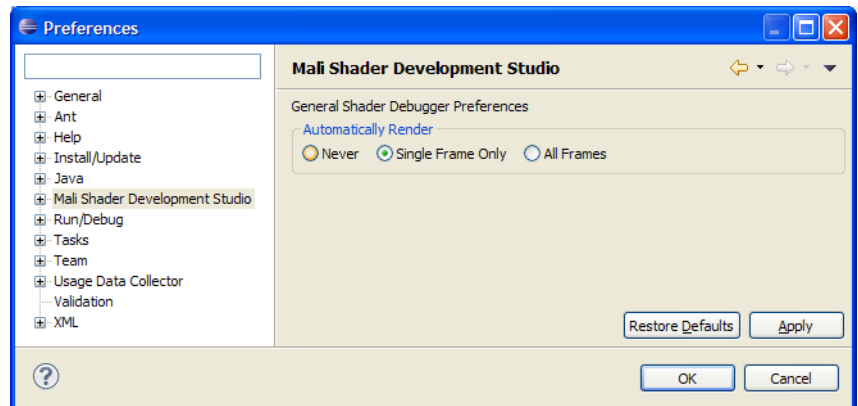




Figure 4-1 Shader Development Preferences pane

#### Automatically Render

The **Automatically Render** group box sets the updating preference for the preview window. The options are:

**Never** Do not automatically render any frames at any point. You can perform manual rendering by clicking on the:

- *Render Single Frame* icon  , or
- *Render All Frames* icon  in the **Shader Control** view.

#### Single Frame Only

Only render the first frame of a shader animation, even if more frames are requested in the **Shader Control** view.

**All Frames** Always render all requested frames.

### 4.1.2 Compiler

This preference pane sets options for the offline shader compiler. The offline shader compiler is used to check shader source files for errors and to report performance statistics of shaders. See Figure 4-2 on page 4-3:

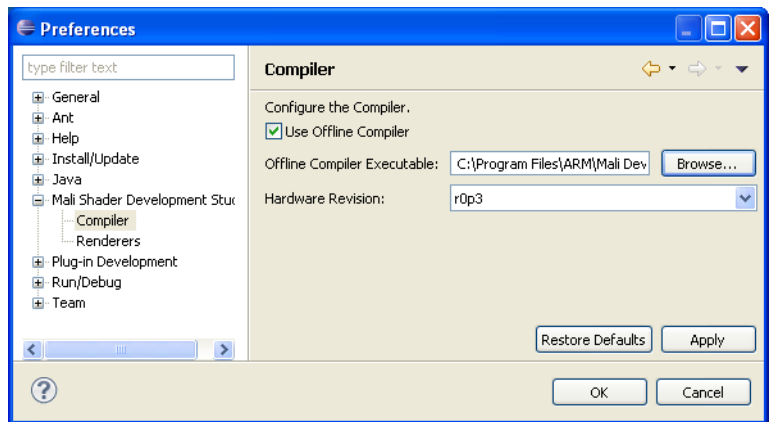


Figure 4-2 Compiler preferences pane

### Use Offline Shader Compiler

If this box is checked, the offline shader compiler is used to check shaders for errors. Unchecking this box disables this behavior.

### Offline Shader Compiler Executable

The location of the offline shader compiler executable. The offline shader compiler is provided as part of the Mali Developer Tools.

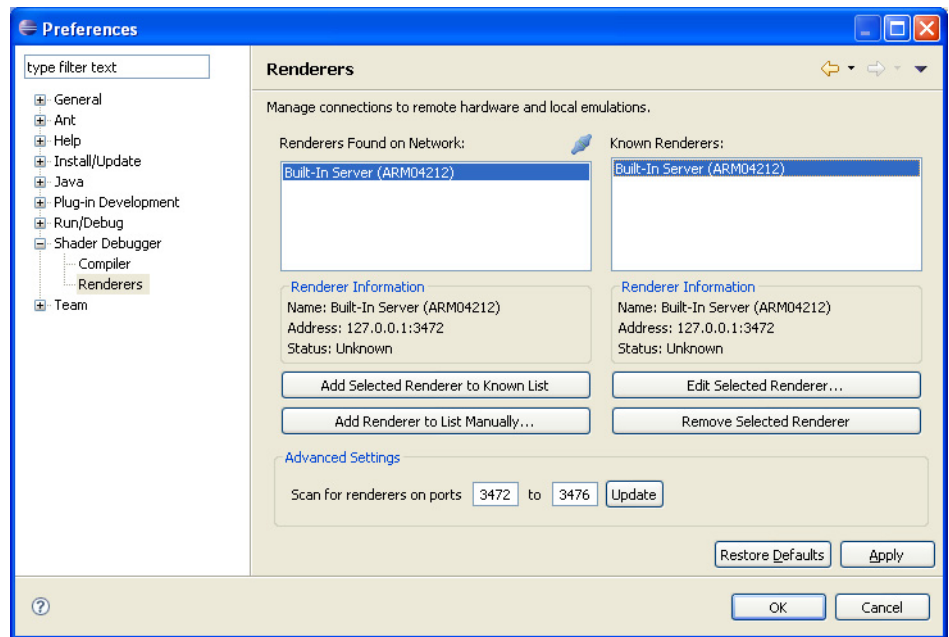
### Hardware Revision

The hardware revision of the Mali GPU to compile shader effects for. If you are not writing shader effects for the latest version of the Mali hardware, set this to the hardware revision you are developing for.

This setting causes the compiler to report revision-specific errors. The **No Specified Revision** option defaults to use the latest version of the Mali hardware. For more information, see the documentation for `maliisc.exe` in the *Mali Offline Shader Compiler User Guide*.

## 4.1.3 Renderers

The **Renderers** pane of the **Preferences** dialog box controls connections to available renderers that are detected on the local network. See Figure 4-3 on page 4-4:



**Figure 4-3 Renderers Preferences pane**

For more information about using the **Renderers** pane, see *Select a renderer* on page 2-11.

### Renderers found on network

This list shows all the available renderers on the current network. The list updates automatically continuously when the preference pane is open.

#### ———— Note —————

For technical reasons the Linux version does not have this capability.

### Known Renderers

This list is a static collection of renderers that have been added during the use of the Shader Development Studio.

### Renderer Information

These two group boxes display information about the currently selected renderer in the **Renderers** lists. The information provided in each group box is:

- Name**        The name of the renderer.
- Address**     The IP address and port number of the renderer.
- Status**       The current status of the renderer.

### Add Selected Renderer to Known List

Add the selected renderer to the **Known Renderers** list.

**Add Renderer to List Manually...**

Opens a dialog box that enables a connection to a renderer to be specified manually, using the following data:

**Connection Name**

Name for the connection, as it appears in the **Known Renderers** list.

**Connection Address**

The IP address of the renderer.

**Connection Port**

The TCP port used to communicate with the renderer. The default connection port is port 3472.

**Edit Selected Renderer ...**

Edit the details of the currently selected renderer.

**Remove Selected Renderer ...**

Remove the renderer currently selected in the **Known Renderers** list.

**Advanced Settings**

This group box specifies the range of ports (inclusive) that are scanned when looking for renderers on the current network. This setting is useful to avoid sending large amounts of discovery data across a network, if you already know the port range of the renderers to use.

Click **Update** to update the scanner to use the ports specified in the **Advanced Settings** group box.

## 4.2 Wizards

This section describes the wizards that are provided by the Shader Development Studio.

### 4.2.1 New Vertex Shader

The New Vertex Shader wizard (accessed from **File** → **New** → **Vertex Shader**) creates a new, trivial vertex shader in the selected parent folder. The file name can be any file name accepted by the host operating system, but ARM recommends that you use the extension `.vert` for vertex shader sources.

For more information about using the New Vertex Shader wizard, see *Creating a new Shader effect* on page 3-3.

### 4.2.2 New Fragment Shader

The New Fragment Shader wizard (accessed from **File** → **New** → **Fragment Shader**) creates a new, trivial fragment shader in the selected parent folder. The file name can be any file name accepted by the host operating system, but ARM recommends that you use the extension `.frag` for fragment shader sources.

For more information about using the New Fragment Shader wizard, see *Creating a new Shader effect* on page 3-3.

### 4.2.3 New ESSL File

The new ESSL File wizard (accessed from **File** → **New** → **ESSL File**) creates a new generic shader file that can be included alongside vertex and fragment shaders in a shader effect. ESSL files are useful for creating `#define` values for use in vertex and fragment shaders.

When using additional ESSL files, you must place them before the vertex and fragment shaders in the shader effect configuration, or the values created in the ESSL file might not be visible in the other files. The compiler concatenates each of these files together, including the available vertex and fragment shaders, before compiling the source.

### 4.2.4 New Shader Configuration

The New Shader Configuration wizard creates a new shader configuration file in the selected project. No other shader configuration file can exist in the project. The wizard creates a shader configuration with a default shader effect, named `New Effect`, that has no shader sources defined in it.

For more information about using the New Shader Configuration wizard, see *Creating a new Shader Configuration File* on page 3-5.

## 4.3 Editors

This section describes the editors that are provided by the Shader Development Studio.

### 4.3.1 Shader editor

The Shader editor provides functionality for editing vertex and fragment shader files. This includes syntax highlighting, error reporting and shortcuts for rendering shader effects.

#### Context Menu

The context menu for the Shader editor provides a shortcut for setting the current shader effect. To display the context menu, right-click in the **Shader editor** view.

#### Render Shader

The Render Shader menu item presents a list of shader effects that reference the current shader source file. Selecting one of these effects causes that effect to become the currently used shader effect in the **Shader Control** view. If automatic rendering is enabled, the **Shader Preview** view is updated to show the effect.

### 4.3.2 Shader Configuration editor

The Shader Configuration editor provides a method for editing a shader configuration file.

For more information about using the Shader Configuration Editor, see *Creating a new Shader Configuration File* on page 3-5.

#### Current Effect

The current shader effect is shown in the **Current Shader Effect** drop-down list. To select a different effect, select it from the list. The other fields in the editor reflect the contents of the currently selected effect.

#### Add New Effect

Creates a new, empty shader effect. A dialog box prompts for a name for the shader effect. Clicking **OK** creates the effect and causes the editor to switch to this effect

#### Rename Effect

Renames the currently selected shader effect. A dialog box prompts for a new name. Clicking **OK** renames the effect.

#### Remove Effect

Removes the shader effect from the configuration.

#### Vertex Shaders section

The vertex shaders section lists all the shader source files that make up the vertex part of the shader effect. It is valid to put a non-specific shader, one that is neither a vertex nor a fragment shader, in this list. But, be aware that the shader is compiled and checked for errors as a vertex shader by the offline compiler. Ordering of source files might be important to the functionality of the shader effect, so buttons are provided to re-order the list.

**Add Shader** Add a new shader source file to the list. The source file must belong to the current project.

**Remove Shader**

Remove the selected shader source file from the list.

**Move Up** Move the selected shader source file up one position. The shader sources are passed to the compiler in the order they appear in the list.

**Move Down** Move the selected shader source file down one position. The shader sources are passed to the compiler in the order they appear in the list.

**Fragment Shaders section**

The fragment shaders section lists all the shader source files that make up the fragment part of the shader effect.

———— **Note** —————

It is valid to put a non-specific shader, one that is neither a vertex nor fragment shader, in this list. But, be aware that the shader is compiled and checked for errors as a fragment shader by the offline compiler. Ordering of source files might be important to the functionality of the shader effect, so buttons are provided to re-order the list.

**Add Shader** Add a new shader source file to the list. The source file must belong to the current project.

**Remove Shader**

Remove the selected shader source file from the list.

**Move Up** Move the selected shader source file up one position. The shader sources are passed to the compiler in the order they appear in the list.

**Move Down** Move the selected shader source file down one position. The shader sources are passed to the compiler in the order they appear in the list.

**4.3.3 Matrix and Vector Editor**

You can use the Matrix and Vector editor to edit shader attribute and uniform values in the Shader Attributes view and Shader Uniforms view. The editor presents a table that can be used to edit the values of vector and matrix OpenGL ES 2.0 types. The drop-down list at the top of the editor enables loading of pre-defined values. The exact functionality available in the editor is dependent on the context it is used in.

**Import** Import values from a Comma Separated Value file.

**Export** Export the current values to a Comma Separated Value file.

For more information about the Matrix and Vector Editor, see *Rendering the shader effect* on page 3-8.

**4.3.4 Sampler Editor**

The Sampler editor is used to edit settings for OpenGL ES 2.0 sampler types (`sampler2D`, `samplerCube`).

**Texture** Select the texture to be used for the sampler.

**Use Built-in** If ticked, use one of the built-in textures for this sampler. Select a texture from the drop down list. If unticked, use the **Browse...** button to select a texture from the project.

The Sampler editor also specifies how OpenGL filters and wraps the texture.

## 4.4 Views

This section describes the different Views that are provided by the Shader Development Studio.

### 4.4.1 Shader Preview

The **Shader Preview** view provides visual feedback about the currently selected shader effect. For information about selecting a shader effect, see *Shader Control* on page 4-11. The current shader effect is displayed at the top of the view. The view is mainly occupied by the Shader preview, that shows a preview of how the currently selected shader effect looks when rendered on the current renderer.

When a preview is being displayed, you can right-click on the preview to save the preview frames as a series of image files.

For more information on using the Shader Preview view, see *Rendering the shader effect* on page 3-8.

#### Animation transport

The animation transport contains a set of buttons to control the animation of shaders that have more than one frame. See Figure 4-4:



Figure 4-4 Animation transport controls

Table 4-1 describes the use of the slider and buttons:

Table 4-1 Animation transport buttons

Button	Description
Slider	The slider at the top shows the position in the current animation, and can also be dragged to move through the animation.
Rewind to Start	Set the current frame to the initial frame.
Play in Reverse	Play from the current frame to the initial frame.
Stop	Stop the animation.
Play Forward	Play from the current frame to the final frame.
Fast Forward to End	Set the current frame to the final frame.

#### Toolbar








The toolbar menu of the Shader Preview view contains a set of buttons that control the current renderer. See Figure 4-5:



Figure 4-5 Toolbar buttons

Table 4-2 describes the toolbar buttons:

**Table 4-2 Toolbar buttons**

Button	Name	Description
	Save Current Frame	Save the currently displayed frame to disk.
	Save All Frames	Save all the frames of the current shader effect to disk.
	Render Single Frame	Render a single frame of the current shader effect.
	Render All Frames	Render all the frames, as specified by <b>Number of Frames</b> in the <b>Shader Control</b> view, of the current shader effect.
	Set Preview Size	Change the size if the preview area.
	Select Default Rendering Mode	Change the default rendering mode. This preference is synchronised with the Automatically Render setting on the Shader Development Studio preference pane, see <i>Automatically Render</i> on page 4-2.
	Select Renderer	Select a renderer to render the current shader effect. This list is populated from the Renderers preference pane, see <i>Renderers</i> on page 4-3. At the end of the list is Manage Renderers. This provides a shortcut to the <b>Renderers</b> preferences page. See <i>Renderers</i> on page 4-3.

### Context menu

The Context menu is displayed by right-clicking over the animation. The buttons on the Context menu are described in Table 4-3:

**Table 4-3 Context buttons**

Menu item	Description
Save Current Frame	Save the currently displayed frame to disk.
Save All Frames	Save all the frames of the current animation to disk.
Set Background Color	Set the color to render the background in the preview. This setting is synonymous with the <code>glClearColor</code> OpenGL ES 2.0 function.

## 4.4.2 Shader Control

The Shader Control view controls the Shader Preview view. It is used for:

- selecting the shader effect to be rendered
- selecting the geometry to render the effect onto
- selecting the number of frames to be rendered
- controlling the camera position for the first and last frame.
- controlling the number of frames to be rendered.

The top of the Shader Control view displays the currently selected shader effect.

For more information on using the Shader Control view, see *Rendering with different geometries* on page 3-12.

## Performance info ...

Displays a dialog box that provides statistics on the performance of the current shader effect. See Figure 4-6:

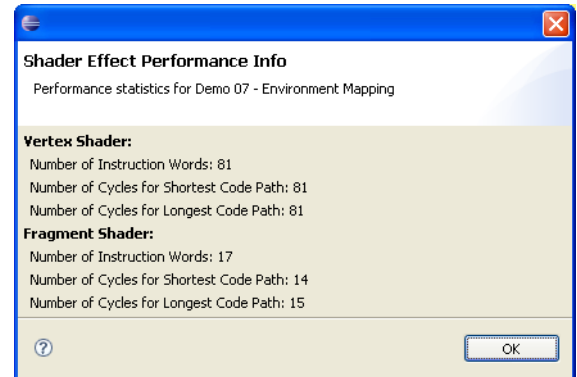


Figure 4-6 Performance Info dialog box

## Camera control

The **Camera control** group box controls the camera positions and settings for rendering shader effects. The settings are described in Table 4-4:

Table 4-4 Camera control buttons

Setting	Description
First Frame	Sets the X, Y, Z and zoom values for the camera in the first frame.
Last Frame	Sets the X, Y, Z and zoom values for the camera in the last frame.
Orthographic Projection	If this box is checked, the geometry is rendered in orthographic projection mode in the <b>Shader Preview</b> view.
Number of Frames	Sets the number of frames to render for shader animation. The values of the uniforms, attributes, and camera values for the shader are interpolated over this number of frames.

## Scene control

The **Scene Control** group box controls how a scene is rendered in the Shader Preview view. The settings are described in Table 4-5:





Table 4-5 Scene control buttons

Button	Description
Enable Blending	Enables the blending capability provided by OpenGL ES.
Configure Blending Options	Display a dialog to configure how OpenGL ES performs blending.
Geometry	Selects the geometry to render the shader effect on.

## Toolbar

The toolbar menu of the Shader Control view contains a set of items that control the current shader effect.

**Table 4-6 Toolbar control buttons**

Button	Name	Description
	Open Source for this Shader	Open any shader source files associate with the currently selected shader effect.
	Reload Shader Configuration	Reload the shader effect's attributes, uniforms, camera and scene control settings from the data stored in the shader configuration.
	Save Settings to Shader Configuration	Write the current shader effect's uniform, attribute, camera control and scene control values back into the shader configuration.
	Select Shader to Render	Click this button to open a menu system that enables navigation to all known shader effects in all open projects. Click on any effect displayed to cause the Shader Preview view to be updated with the selected shader effect. The automatic rendering preference is used to determine if the selected shader is rendered. See <i>Automatically Render</i> on page 4-2.

### 4.4.3 Shader Attributes

The Shader Attributes view presents a list of all attributes in the current shader effect. This list is generated from the data in the shader effect.

For more information about using the Shader Attributes view, see *Rendering the shader effect* on page 3-8.

Table 4-7 describes the shader attributes that are displayed in the Shader Attributes view:

**Table 4-7 Shader Attributes**

Attribute	Description
Loc	The location of the attribute in the shader, as assigned by the renderer.
Name	The name of the attribute as it appears in the shader source file.
Type	OpenGL ES Type of the attribute.
Value	Value of the attribute. Click on this cell to present an editor appropriate to the type of the attribute. For example, vectors and matrices present a table editor, though for single-value types such as float, the current cell is edited.

### 4.4.4 Shader Uniforms

The Shader Uniforms view presents a list of all uniforms in the current shader effect. This list is generated from the data in the shader effect.

For more information about using the Shader Uniforms view, see *Rendering the shader effect* on page 3-8.

Table 4-8 describes the shader uniforms that are displayed in the Shader Uniforms view:

**Table 4-8 Toolbar control buttons**

<b>Button</b>	<b>Description</b>
Loc	The location of the uniform in the shader, as assigned by the renderer.
Name	The name of the uniform as it appears in the shader source file.
Type	The OpenGL ES Type of the uniform.
Initial Value	The value of the uniform for the first frame. For animation, this is the initial value the uniform values are interpolated from. For a single frame shader effect, this sets the value of the uniform for that frame. Clicking on this cell presents an editor appropriate to the type of the attribute.
Final Value	The value of the uniform for the last frame. For animation, this is the final value the uniform values are interpolated to. For a single frame shader effect, this has no effect. Clicking on this cell presents an editor appropriate to the type of the attribute.

# Appendix A

## Shader Server

This appendix describes how to use a remote renderer for the Shader Development Studio. It consists of the following section:

- *About the Shader Server* on page A-2.

## A.1 About the Shader Server

Shader Server is a remote renderer for the Shader Development Studio. It accepts Mali Remote Interface and OpenGL ES 2.0 calls over the network, executes those calls on Mali-200 or Mali-400 hardware, and returns the results to the client.

It consists of two parts:

- A library implementing the communication protocol and function dispatch, called `libmri`.
- A remote implementation of the function calls, that interfaces with OpenGL ES 2.0 and the operating system.

The source code provided is designed to be built for an ARM based Linux operating system, with Mali-200 or Mali-400 OpenGL ES 2.0 drivers.

This section describes:

- *Requirements*
- *Contents*
- *Building* on page A-3
- *Usage* on page A-4
- *Security* on page A-4
- *Porting* on page A-5.

### A.1.1 Requirements

The Shader Server source code is provided as part of the Shader Development Studio. It is located in the `ShaderServer` directory under the installed path of the Shader Development Studio.

To build the Shader Server you require:

- a recent GNU compiler toolchain targeting the ARM architecture, including:
  - C++ compiler,
  - linker
  - archiver
  - Make.
- an OpenGL ES 2.0 driver to link against.

The Shader Server has been successfully built and tested on a computer with:

- Fedora 8 (32-bit) with Linux kernel 2.6.25.11-60.fc8
- ARM GCC cross compilation toolchain 4.2.0 20070413
- Make 3.81.

### A.1.2 Contents

The source code consists of:

- `readme.txt`
- `Makefile`
- `libmri/`
  - `glesfunctiontable.h`
  - `gleslocalfunctions.c`

- gleslocalfunctions.h
- Makefile
- mrp.h
- mrpclient.cpp
- mrpclient.h
- mrperror.cpp
- mrperror.h
- mrpmessagedecoder.cpp
- mrpmessagedecoder.h
- mrpmessageelement.cpp
- mrpmessageelement.h
- mrpmessageencoder.cpp
- mrpmessageencoder.h
- mrpnetio.cpp
- mrpnetio.h
- mrpserver.cpp
- mrpserver.h
- mrpstring.cpp
- mrpstring.h
- Shader Server/
  - dogles\_remote\_implementation/
    - bufferhandling.cpp
    - drawingcontrol.cpp
    - logging.cpp
    - peermanagement.cpp
    - performance\_counters.cpp
    - renderingcontrol.cpp
    - state.cpp
    - state.h
  - Mali200/3rdparty/include/khronos/EGL/eglplatform.h
  - Mali200/3rdparty/include/khronos/EGL/fbdev\_window.h
  - Mali200/3rdparty/include/khronos/KHR/khrplatform.h
  - Makefile
  - ShaderServer.cpp
  - stafx.cpp
  - stafx.h

### A.1.3 Building

To build the shader for the Mali-200 reference platform:

1. Go to the installed location for the Shader Development Studio.
2. Change to the Shader Server directory.

3. Download the following Khronos OpenGL ES 2.0 header files from the Mali Developer Center website at: <http://www.malideveloper.com>
  - GLES2/gl2.h
  - GLES2/gl2ext.h
  - GLES2/gl2platform.h
4. Copy gl2.h, gl2ext.h, and gl2platform.h to:  
*installation\_directory/ShaderServer/MALI200/3rdparty/include/khronos/GLES2*
5. Download the following Khronos EGL header files from the Mali Developer Center website at: <http://www.malideveloper.com>
  - egl.h
  - eglxext.h
6. Copy egl.h and eglxext.h to:  
*installation\_directory/ShaderServer/MALI200/3rdparty/include/khronos/EGL*
7. In the files libmri/Makefile and ShaderServer/Makefile, modify the MALI\_LIB\_DIR variables to point to the Mali-200 or Mali 400 OpenGL ES 2.0 driver libraries.
8. Change to the root level, that is the directory that contains libmri and ShaderServer, and enter make.
9. The source code builds and produces an executable called shaderserver in the ShaderServer directory. You can copy this to your ARM development platform.

#### A.1.4 Usage

Start the Shader Server with argument `--help` for usage information.

The Shader Server requires a port and a server name as arguments:

```
shaderserver -p port -n name [-l] [-v]
```

where:

- |             |   |
|-------------|---|
| <i>port</i> | is the network port number on the platform's network interface used to listen for incoming connections. Shader Development Studio searches for servers between ports 3472 and 3476, by default. |
| <i>name</i> | is an identifier string that is displayed by the Shader Development Studio when it discovers the server.  |
| -l          | If present, this switch causes the server to listen for connections from only the localhost (127.0.0.1).  |
| -v          | If present, this switch turns on verbose output for debugging.  |

#### A.1.5 Security

Running Shader Server permits any remote user to connect to your platform and execute OpenGL ES 2.0 code with the privileges of the user that the server is run by.

It must be considered an INSECURE service, and must only be made accessible from a trusted network or through VPN or *Secure Shell* (SSH) tunneling.

## A.1.6 Porting

The Shader Server depends on the following components:

- C library functions for example `malloc`, `free`, and `string` operations. These are called through `#defines`, found at the beginning of `mrp.h`.
- BSD style network socket functions for example, `bind`, `listen`, `accept`, `send`, and `recv`. Calls to these functions are found in the `mrpclient.cpp`, `mrpserver.cpp` and `mrpnetio.cpp`.
- OpenGL ES 2.0 functions. These are called through a generic function call mechanism in `mrpserver.cpp`. The list of available functions is defined in the array of structures in `glesfunctiontable.h`.

Porting the code to an alternative operating system or platform requires these three components to be provided or emulated.

# Glossary

This glossary describes some of the terms used in this document.

<b>Debugger</b>	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.
<b>ESSL</b>	<i>See</i> OpenGL ES Shading Language.
<b>ETC</b>	<i>See</i> Ericsson Texture Compression.
<b>Fps</b>	Frames per second. The number of individual frames that are rendered every second, to create an animation. Typical games and GUI applications run at 20-30 fps.
<b>Fragment shader</b>	A program running on the pixel processor that calculates the color and other characteristics of each fragment.
<b>Framebuffer</b>	A memory buffer containing a complete frame of graphical data.
<b>Geometry processor</b>	A geometry processor, such as the MaliGP2, executes vertex shaders that typically contain transform and lighting calculations, and generates lists of primitives for a pixel processor to draw.
<b>GPU</b>	<i>See</i> Graphics Processor Unit.
<b>Graphics application</b>	A custom program that runs on the platform CPU and uses the Mali GPU to display graphics.
<b>Graphics driver</b>	A software library implementing OpenGL ES or OpenVG, using graphics accelerator hardware. See also OpenGL ES driver and OpenVG driver.

**Graphics Processor Unit (GPU)**

A hardware accelerator for graphics systems. Mali programmable GPUs, such as the Mali-400 GPU and the Mali-200 GPU, consist of a geometry processor and one or more pixel processors. Mali fixed-function GPUs, such as the Mali-55 GPU, consist of a pixel processor only.

**JAR**

Java Archive. (For more information, see <http://java.sun.com>.)

**Mali**

A name given to graphics software and hardware products that aid 2D and 3D acceleration.

**Mali Developer Tools**

A set of development programs that enables software developers to create graphic applications.

**Offline Shader Compiler**

The `malisc.exe` offline shader compiler is provided with the Mali Developer Tools and is used to provide support functionality to the Shader Development Studio, for example, by compiling shader effects in the background and reporting any errors. The offline shader compiler is a command line tool that translates vertex shaders and fragment shaders written in the ESSL into binary vertex shaders and binary fragment shaders that you can link and run on the GPU.

**OpenGL ES driver**

On graphics systems that use the OpenGL ES API, the OpenGL ES driver is a specialized driver that controls the graphics hardware.

**OpenGL ES Shading Language (ESSL)**

A programming language used to create custom shader programs that can be used within a programmable pipeline, on graphics hardware. You can also use pre-defined library shaders, written in ESSL.

**Pixel**

A pixel is a discrete element that forms part of an image on a display. The word pixel is derived from the term Picture Element.

**Red Green Blue Alpha (RGBA)**

An implementation of the RGB color model, that includes an Alpha value to indicate opacity. If a fragment has an Alpha value of 0%, it is fully transparent, making it invisible. An Alpha value of 100% results in a fully opaque pixel.

**Render**

Rendering in the context of the Shader Development Studio plug-in refers to the sending of shader effects to a remote Renderer, so that a preview image can be generated.

**Renderer**

A device capable of rendering shader effects and returning an image of the rendered shader through using the Mali Remote Protocol. Typically, this is a device with a Mali GPU, though it can also be a local or remote emulation.

**Shader**

A Shader or Shader Source file, refers to a single source file for a shader effect. This source file might be a vertex shader, a fragment shader or a non-specific utility shader that can be linked concatenated and compiled with other shaders.

**Shader configuration**

A Shader configuration is a container for shader effects. The shader configuration is defined in a file named `shaders.shaderconfig`. Only one of these files can be defined in a single project, and that file must exist at the project root, that is, at the top level of the project.

**Shader Library**

A set of shader examples, tutorials, and other information, designed to assist with developing shader programs for the Mali GPU. The Shader Library is a component of the Mali Developer Tools.

**Shader Effect**

A Shader effect is a collection of one vertex shader, one fragment shader and zero or more non-specific shaders that make a complete effect. This is the smallest unit that can be rendered.

**Vertex shader**

A program running on the geometry processor, that calculates the position and other characteristics, such as color and texture coordinates, for each vertex.