Freescale Semiconductor
Application Note

Document Number: AN3793
Rev. 0, 12/2008

3D Graphics on the ADS512101 Board Using OpenGL ES

by: Francisco Sandoval Zazueta
Infotainment Multimedia and Telematics (IMT)

1 Introduction

OpenGL is one of the most widely used graphic standard specifications available. OpenGL ES is a reduced adaptation of OpenGL that offers a powerful yet limited version for embedded systems.

One of the main features of the MPC5121e is its graphics co-processor, the MBX core. It is a wide spread standard of mobile 3D graphics acceleration for mobile solutions. Together with Imagination Technologies OpenGL ES 1.1 SDK, the ADS512101 board can be used to produce eye-catching graphics. This document is an introduction, after the development environment is ready, it is up to the developer's skills to exploit the board's graphics capabilities.

Contents

freescale semiconductor

This document guides the reader through the required process to set up a development environment with OpenGL ES (OGLES) using an ADS512101 board and LTIB. This includes configuring the board's display interface unit (DIU), installing the required files and libraries in both the board and host computer, configuring the OGLES SDK, and creating an application that loads a complex 3D model.

After the completion of this application note, the reader will have the required knowledge to use OGLES in applications written for the ADS512101 board, and will be familiar with OGLES basics.

2 Preparing the Environment

2.1 Assumptions on the Environment

Start with a new Linux target image builder (LTIB) installation. The ADS512101 board must be properly booting to Linux using a network file system (NFS). The user must have access to the board via the serial terminal. It is important to have a basic knowledge of LTIB and the ADS512101 board. Many documents are available online for the correct BSP configuration.

This document was created using the following versions of hardware and software:

- ltib-mpc5121ads-20080528 with the patches bundled in this application note. Refer to Appendix A, " Applying DIU Patches to the Kernel" for information on applying the patches.
- ADS512101 Rev 3.2 board that uses a rev 1.5 of the MPC5121e silicon
- MPC5121 ADS MBX OpenGL ES driver Build09, bundled in this application note
- Host system running Ubuntu 8.04 (Hardy Heron)
- Linux MPC5121 MBX OpenGL ES 1.1 SDK [2.03.23.0793] is in the software that goes with this application note, AN3793SW

If using anything different, the procedure may be different but this document can continues serving as a general guide.

2.2 Linux Target Image Builder (LTIB)

2.2.1 U-boot Display Interface Unit (DIU) Support

Make sure the u-boot supports the DIU. For this use the u-boot generated by the LTIB. Run the LTIB (`./ltib`) to build the BSP, then complete the following steps:

1. After the LTIB installation, copy the install script supplied here to the LTIB installation directory (referred from now on as `<ltib_installation_folder>`)and run it:
   - `ubuntu#cp install (<ltib_installation_folder>)/`
   - `ubuntu#chmod 777 install`
   - `ubuntu#sudo ./install`
   - `ubuntu#cd (<ltib_installation_folder>)`
   - `ubuntu#sudo ./install`

2. This script copies the files generated by the LTIB (uImage, u-boot.bin, and mpc5121ads.dtb) to the tftp folder (in this case /tftpboot/), creates a dynamic link in this folder to the LTIB's rootfs, then restarts the NFS server. Please modify accordingly if the LTIB's folder structure and the tftp folder differ.
3. From this point execute the following instruction from u-boot on the ADS512101 board.
4. Update the board's u-boot to the new one generated by the LTIB. When properly configured, this can be achieved by typing the following command: `-sh-2.05b#/=>run upd.` If difficulties arise, please refer to the ADS512101 board documentation.
5. Make sure the ADS512101 DVI output is connected to a display.
6. Re-start the board, then boot to Linux using NFS `run net_nfs.` In the display the Freescale logo appears and then a penguin. Otherwise, follow the instructions to configure the DIU under Appendix C, "Configuring DIU Kernel Modules".

2.2.2 MBX Libraries

Applications running on the board need to call MBX libraries to use the core and perform operations needed to render 3D graphics. To achieve this, the PowerVR services have to be running in Linux.

Copy the MBX_libraries.tar.gz file to `(<ltib_installation_folder>)/rootfs/`, and then extract it respecting file paths.

Boot up the board and then execute the following command:

```
-sh-2.05b#/etc/init.d/rc.pvr start
```

The following messages must appear:

```
[   38.129732] dbgdrv: module license 'unspecified' taints kernel.
[   38.379928] CLCDC_Init: major device 251
[   38.385066] ============================================================
[   38.393824] * MBX Driver, ALT Software Inc. *
[   38.402712] * Build 05, Multi-plane support, March 4, 2008 *
[   38.411433] ============================================================
[   38.420131] Setting up driver for:                  fb0
[   38.424826] DIU Framebuffer start:                  0x 4501000
[   38.430173] DIU Framebuffer virtual:         0xc4501000
[   38.435584] DIU Framebuffer size:                   3145728 bytes
[   38.441013] ============================================================
[   38.449761] bits_per_pixel:    32
[   38.453854] width:             1024
[   38.458063] height:            768
[   38.462268] red.length:        8
[   38.466263] green.length:      8
[   38.470213] blue.length:       8
[   38.474723] AllocContiguousMemory: pLinAddr: dec00000 -- dma: 1ec00000 -- siz
e: 3145728
[   38.483935] AllocContiguousMemory: pLinAddr: de800000 -- dma: 1e800000 -- siz
e: 3145728
[   38.605424] CAMERA_Init: major device 250
[   38.610264] AllocContiguousMemory: pLinAddr: df180000 -- dma: 1f180000 -- siz
e: 524288
[   38.620333] AllocContiguousMemory: pLinAddr: df200000 -- dma: 1f200000 -- siz
e: 524288
```

```
[   38.629166] AllocContiguousMemory: pLinAddr: df280000 -- dma: 1f280000 -- siz
e: 524288
Loaded PowerVR consumer services.
```

NOTE

If after running `/etc/init.d/rc.pvr start` a fatal error appears referring to the fsl_diu_flip file, this is because LTIB version 20080528 or older is running. To apply the proper patches refer to Appendix A, " Applying DIU Patches to the Kernel".

2.3 Installing PowerVR Software Development Kit (SDK)

2.3.1 Preparing SDK

The PowerVR software development kit (SDK) is used to program applications using OGLES libraries. The next steps are used to prepare and configure the SDK.

Extract the OpenGL-ES_SDK_5121.tgz file in the host machine then complete the following steps:

1. Move the extracted folder (SDKPackage) to the desired location. For this example use: `/home/fslserver/SDKPackage`.
2. Copy the OGLES libraries to …`/SDKPackage/Builds/OGLES/LinuxMPC512/Lib/`. The OGLES libraries are located in the MBX_libraries.tar.gz file under `/usr/lib/.`
3. Verify the SDKPackage/Builds/OGLES/LinuxMPC512/make_platform.mak file that points to the MPC5121's toolchain. For example:

```
ifndef TOOLCHAIN
TOOLCHAIN=/opt/freescale/usr/local/gcc-4.1.78-eglibc-2.5.78-1/powerpc-e300c3-linux-gnu
endif
ifndef CROSS_COMPILE
ifndef CROSS_COMPILE_PREFIX
CROSS_COMPILE = $(TOOLCHAIN)/bin/powerpc-e300c3-linux-gnu-
else
CROSS_COMPILE = $(TOOLCHAIN)/bin/$(CROSS_COMPILE_PREFIX)
endif
endif
```

4. The SDK is ready to be used.

For further information refer to the `OpenGL ES SDK.User Guide.1.1f.External.pdf` file found in the SDK.

2.3.2 Compiling and Running Sample Application

To compile a sample application, complete the following steps:

1. Open a host console window, then switch to the following path under the SDK directory:
2. `../SDKPackage/TrainingCourse/02_HelloTriangle/OGLES/Build/LinuxMPC512/`
3. Type the following command — `ubuntu#make PLATFORM=LinuxMPC512 Common=1`
4. Switch to — `../OGLES/Build/LinuxMPC512/Common/Release`

5. Copy the binary file in the NFS to `<rootfs>/usr/local/bin`
6. Boot up the board and start the PowerVR services — `-sh-2.05b#/etc/init.d/rc.pvr start`
7. On the board, switch to /usr/local/bin/, then run the file:
   - `-sh-2.05b#cd /usr/local/bin`
   - `-sh-2.05b#./OGLESHelloTriangle`
8. A triangle in the display connected to the board's DVI output then appears.
   - To close the demo application, use Ctrl+c

3 The PowerVR SDK

3.1 Introduction to SDK

The POWERVR MBX family of IPcores is fully compliant with the Khronos OpenGL ES 1.x API. The MPC5121 SDK includes tutorials, source code, documentation, extensions specifications, platform abstraction framework, and a tools library to create applications with 3D graphics use.

NOTE

If you have no experience using OpenGL ES, read Appendix B, "Brief Introduction to OpenGL ES" before continuing with this application note.

3.2 PVRShell

The OpenGL ES needs a context to render graphics. How the context is obtained has little to do with graphics programming and differs from platform to platform. Khronos defines an Embedded-System Graphics Library (EGL), an interface between its APIs, and the underlying native platform windows system (Windows, X11, and so on).

The PVRShell is a framework that allows platform abstraction. Applications using it work on any MBX enabled device. It manages all OS specific initialization code, and is convenient for writing portable applications. It also has several built in command line features that allow to define attributes, for example the back buffer size, vsync, and anti-aliasing modes.

The PVRShell uses standard EGL and OGLES calls to manage context creation, keypad input, command line parameters, and other functions.

When using this framework, the application must be defined as a class that inherits from the PVRShell, and must implement the following five methods. At execution time these five methods are called in the listed order.

1. InitApplication — This is called before any OGLES initialization has taken place, and can be used to set up any application data that does not require OGLES calls, for example object positions or arrays containing vertex data before they are uploaded.
2. InitView — This is called after OGLES has initialized and can be used to do any remaining initialization that requires API functionality.

3. RenderScene — This is called repeatedly to draw the geometry. Returning false from this function instructs the application to enter the quit sequence.
4. ReleaseView — This function is called before OGLES is released and is used to release any resources allocated by it.
5. QuitApplication — This is called last after the API has been released and can be used to free any leftover user allocated memory.

The shell framework starts the application by calling a NewDemo function that must return an instance of the PVRShell class defined.

For more information, refer to the SDK training 03: Introducing PVRShell (`.../SDKPackage/TrainingCourse/03_IntroducingPVRShell`), as well as its documentation (`.../SDKPackage/Shell/Documentation`).

3.3 PVRtools

The PVRTools is a collection of source code to help developers with common tasks that are frequently used in 3D programming. The PVRTools supply code for mathematical operations, matrix handling, loading 3D models, and optimizing geometry. The API specific tools contain code for displaying text and loading textures.

To learn more about the included tools, refer to the SDK tool documentation.

3.3.1 PVRTexTool

The SDK also contains a collection of utilities to help developers import textures and models to applications. For example, the demo bundled with this application note uses the command line PVRTexTool for Linux (`…/SDKPackage/Utilities/PVRTexTool/PVRTexToolCL/Linux`).

The PVRTexTool is a utility to process and compress textures. To use textured models within the application, the SDK supplies this application to optimize bitmap texture files for use. It either generates a header (.h) or a binary (.pvr) file. For an example, refer to the object loader demo code in the example application of this document.

For more information regarding the included utilities, refer to the SDK online documentation.

4 Developing Example Application

This chapter describes how a complex 3D model is loaded to an application running on the ADS512101 board. This example helps to further understand the process needed to compile and deploy applications using the OGLES libraries.

4.1 3D Model Loader

When working with OGLES, it is easier to design models using a 3D modeling tool (3ds Max or Blender), then loading them in the application.

A .obj loader is included with this application note to land the concepts and tools previously explained and provide the reader with the starting point of a useful tool to import 3D models.

4.1.1 Project Folder Structure

When extracting the objectLoader.zip file, the following tree appears:

```
/objectLoader.cpp                  Sample code using the model loader
/tObjModel3d.cpp                   .cpp file for the model loader library
/tObjModel3d.h                     Header file for the model loader library
/Build/LinuxGeneric/Makefile       The Makefile to compile the project
/DemoFiles/                        Textures, textured and untextured models.
```

Take time to explore the files and become familiar with its content. For the Makefile to work, it is important to preserve this directory structure and set its SDKDIR variable with the path to the installed and correctly configured SDK.

4.1.2 Compiling and Running the Object Loader

1. To compile the source code, open a terminal in the host computer and switch to the `objectLoader/Build/LinuxGeneric` folder, and make for the LinuxMPC512 platform choosing the common profile, because the MPC5121e has the ability to support floating point:
   - `ubuntu#make PLATFORM=LinuxMPC512 Common=1`
2. Copy the generated binary file to NFS:
   - `ubuntu#cd objectLoader/Build/LinuxMPC512/Common/Release/`
   - `ubuntu#sudo cp objectLoader <ltib_installation_folder>/rootfs/usr/local/bin`
3. Then copy the objects and texture included in `/objectLoader/DemoFiles` to the same location in the NFS:
   - `ubuntu#cd objectLoader/DemoFiles`
   - `ubuntu#sudo cp * <ltib_installation_folder>/rootfs/usr/local/bin`
4. Now boot up the board, start the PowerVR services, switch to the `/usr/local/bin` folder, then execute the binary file:
   - `-sh-2.05b#/etc/init.d/rc.pvr start`
   - `-sh-2.05b#cd usr/local/bin`
   - `-sh-2.05b#./objectLoader`
5. A textured box and an untextured skateboard rotating in the display connected to the DVI output of the board must appear. Press q to exit the demo application. If the colors are wrong most likely an LTIB version 20080528 is being used, see Figure 1. A patch must be applied to correct the colors. Please refer to Appendix A, “ Applying DIU Patches to the Kernel” for instructions to solve this issue.

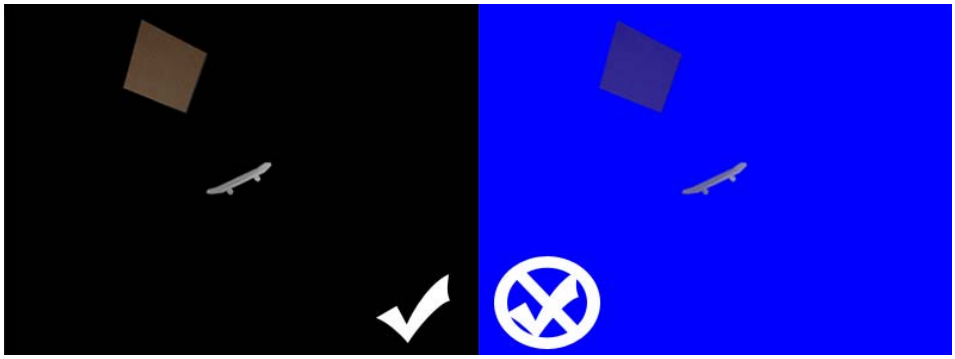

Figure 1. Running Demo

4.1.3 How the Object Loader Works

A .obj file defines the properties of a 3D object including its geometry. It is an open file format developed by Wavefront Technologies and is supported by a multitude of 3D graphic applications. It is in ASCII format and is easily readable with a regular text editor. For more information regarding this file type, open the .obj objects bundled with this application note and refer to the file specification at its web page.

The tObjModel3d class parses a .obj file and loads volatile vectors to the memory with the object properties. This class also provides a method that takes these vectors as input and draws the object using OGLES calls. In the objectLoader.cpp program, two tObjModel3d objects are created; a textured crate (`cube.obj`) and an untextured skateboard (`skateboard.obj`).

Using the PVRShell framework, this application initializes the objects, renders them on-screen, and when closing the application releases the texture used memory.

Please read the code comments to get a better idea of how each method works.

4.1.4 Experimenting with Code

4.1.4.1 Loading New Objects

After the new .obj file (and .pvr texture if it has one) is created, copy it to the NFS folder where the binary files reside.

Create new instances of the tObjModel class passing the .obj file name to its constructor:

```
tObjModel3d object("object.obj");
```

If the object has a texture, modify the if statement in the InitView() method accordingly:

```
if (loadTextures(&object) && loadTextures(&object2)){}
else {//code for error handling};
```

Again, if loading a textured object, release its textures in the ReleaseView() method:

```
glDeleteTextures(1,object.getTextureHandle());
```

The object is ready to be used in the main rendering loop. In RenderScene():

```
object.drawObject();
```

NOTE

Even though creating new objects and textures is out of the scope of this application note, included in the code are comments with hints on how to achieve this.

4.1.4.2 RenderScene() Loop

This is the loop where all the OGLES drawing functions are called. By using matrix transformations move the object around the screen and achieve simple animations. Feel free to experiment with different OGLES calls, for example:

```
glTranslatef(float x, float y, float z); // Multiplies the current matrix by the specified
translation matrix.

glScalef(float x, float y, float z); // Scales the current matrix by the specified
escalation matrix.

glRotatef(float angle, float x, float y, float z); //rotates the current matrix by the
specified rotation matrix.
```

Notice that each matrix operation is applied to the current matrix in the stack, this is why translating then rotating an object is different than rotating then translating it. It is a good practice to use `glPushMatrix()` and `glPopMatrix()` to preserve the previously used model view matrix. It is not recommended to load and operate on the projection matrix if you are not sure of what you are doing.

For more information regarding OGLES and how it works, refer to Appendix B, “Brief Introduction to OpenGL ES”.

5 Conclusion

Even if 3D graphics are usually associated with games, its fields of application have greatly broadened in the last years. 3D graphics are now found in graphic user interfaces, complete motion pictures, user guides, interactive presentations, and cellular phones to mention some examples.

The OpenGL ES can be used to enhance many application presentations. Together with the MPC5121e MBX module, high quality 3D graphics that are hard to achieve by a microprocessor alone can enhance graphic user interfaces and programs.

6 References

The following are a few useful references regarding 3D modeling, OGLES, and other helpful subjects.

- The OpenGL ES 1.1 reference pages are useful when developing OGLES applications
- A book on embedded 3D graphics:
  - *Mobile 3D graphics with OpenGL ES and M3G*, by Kari Pulli, Tomi Aarnio, Ville Miettinen, Kimmo Roimela, Jani Vaarala. Published by Morgan Kaufmann publishers.
- Both Blender (a full-featured open source 3D modeling software) and OpenGL have great communities online. Plenty of information on 3D programming and modeling can be found on the web

7 Glossary

This is a compilation of commonly used abbreviations mentioned in this document.

OGLES — Open Graphics Library Embedded Systems. This is the API used for 3D graphics programming using the MPC5121e MBX.

LTIB — Linux Target Image Builder. The tool used for configuring and deploying the ADS512101's BSP.

NFS — Network File System. The remote file system used for easier application compiling, deploying, and executing.

ADS512101 — Advanced Development System for MPC5121e. This is the board used as the development system.

PVR MBX — PowerVR MBX. A graphics processing unit of 2D/3D accelerator families. The 3D graphics accelerator module of the ADS512101 board.

TFTP — Trivial File Transfer Protocol. The protocol used by u-boot to transfer the kernel to be used by the ADS512101 board.

Khronos — An industry consortium, creators of a number of standards; OpenGL, OpenGL ES, COLLADA, and others.

EGL — Embedded-System Graphics Library. An interface between OGLES and the platform's operating system used to create rendering surfaces.

u-boot — The boot loader generated by LTIB used in this guide for the ADS512101 board.

DIU — Display Interface Unit. The ADS512101's display controller.

BSP — Board Support Package. An implementation specific support code for a given board that conforms to a given operating system.

Appendix A
Applying DIU Patches to the Kernel

A.1 Applying the MBXpatch2.patch

In the LTIB installation, copy the MBXpatch2.patch and diu_flip.patch files to (ltib installation folder)/rpm/SOURCES and the kernel-2.6.24.2-mpc5121.spec.in file to (ltib installation folder)/dist/lfs-5.1/kernel.

- Then prepare, build, and deploy the kernel with:

```
./ltib -p kernel -m prep
./ltib -p kernel -m scbuild
./ltib -p kernel -m scdeploy
```

- Run the install script again. The PowerVR services can now be started now with:

```
-sh-2.05b#/etc/init.d/rc.pvr start
```

A.2 byte_flip Application

The MBX pixel format uses a blue, green, red, alpha (BGRA) byte format and renders exclusively to the framebuffer 0 (/dev/fb0). This application switches the selected framebuffer's pixel format to be compatible with the MBX. The colors are then correctly displayed.

- Copy the byte_flip binary to `(ltib installation folder)/rootfs/bin/`

```
sudo cp byte_flip (ltib installation folder)/rootfs/bin/
```

- Usage — Boot to the board and then from the LTIB prompt:

```
-sh-2.05b#byte_flip fbdevice state
```

The fbdevice can be substituted by any available framebuffer and the state can be either on or off, for example:

```
-sh-2.05b#byte_flip /dev/fb0 off
```

When selecting fb0, the screen color switches from a bright blue to black. This means both the patches and the application are working.

Appendix B
Brief Introduction to OpenGL ES

B.1 OpenGL ES

OpenGL ES is a royalty-free cross platform API for full-function 2D and 3D graphics on embedded systems. It consists of well-defined subsets of desktop OpenGL creating an interface between software and graphics acceleration. There are several versions, but the one supported by the ADS512101 is OpenGL ES 1.1, based on the OpenGL 1.5 specification. Not every embedded device supports a hardware floating point unit, therefore two different profiles are supported: common (CM) and common lite (CL). The difference between the two versions is the omission of all floating point entry points in the CL profile. The MPC5121e uses CM.

OGLES operates as a state machine. Each state consists of various features that are either enabled or disabled. For example, to demonstrate lighting in a scene, both lighting and at least one light must be enabled:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

After a feature is enabled, properties can be adjusted via various functions. To continue with the previous example, do the following; define `LIGHT0` as a spot light and configure its position, :

```
glLightfv(GL_LIGHT0, GL_POSITION, kSpotLight);
glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, 25.f);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, kSpotDirection);
```

The light's position and direction are stored in the kSpotLight and kSpotDirection arrays. Many other properties can be adjusted. For a complete list of features and properties, refer to the OpenGL ES 1.1 Reference Pages.

B.2 Main Differences Between OGLES 1.1 and OpenGL

OGLES is created for embedded systems. One of the main goals was to keep its implementation as compact as possible. A lot of unnecessary functionality from OpenGL is removed. It can only render triangles, lines and points. The number of data formats an API call accepts is reduced to a single integer or single floating point variant (`glRotatex(), glRotatef()`). Features hard to emulate in the application code were retained and convenience features were removed.

OGLES draws geometric objects using exclusively vertex arrays. Associated colors, normal, and texture coordinates are also specified using arrays. OpenGL's `begin` and `end` commands are not supported.

The drawing of quads polygons is not supported. The only supported primitives are triangles, lines, and points as lists, strips, or fans.

Please refer to the sample demo applications provided and trainings bundled with the SDK for coding examples using EGL, OGLES, and PVRShell.

B.3 Obtaining Frustum Numbers

While glOrtho multiplies the current matrix with an orthographic matrix the glFrustum multiplies the current matrix with a perspective. This is easier to understand by looking at the following picture of a cube using both perspective (frustum) and orthographic projections:

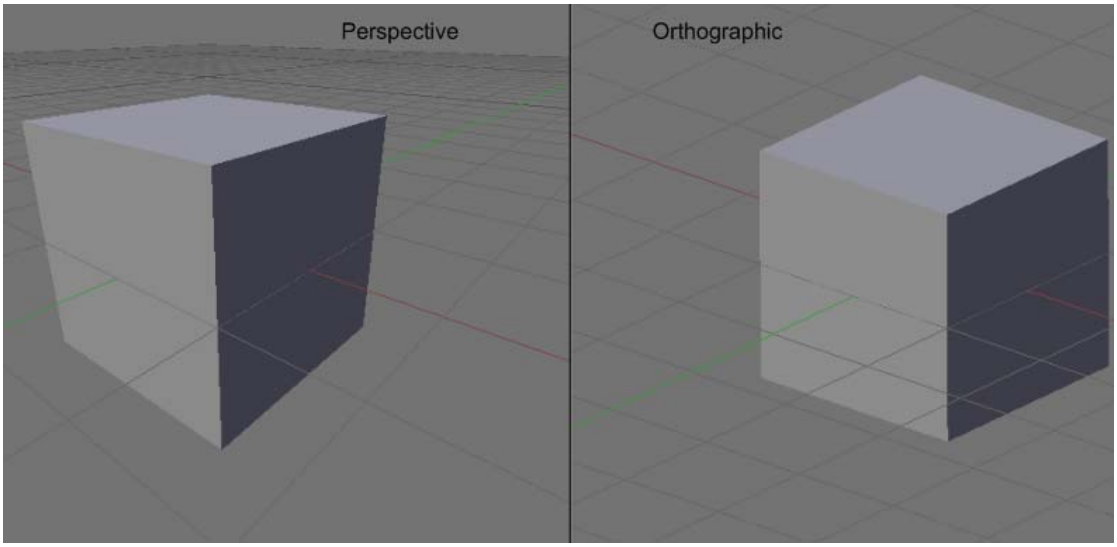

Figure 2. Perspective (frustum) and Orthographic Projections

To achieve a sense of depth, the coordinate system used to represent objects in the objectLoader demo is multiplied by a perspective matrix using glFrustumf().

B.4 glFrustum

The glFrustum function receives the following parameters:

```
void glFrustumf(GLfloat left,
                GLfloat right,
                GLfloat bottom,
                GLfloat top,
                GLfloat near,
                GLfloat far);
```

In the objectLoader.cpp file, the following parameters are used:

```
static const GLfloat kLeft = -10.f;
static const GLfloat kRight = 10.f;
static const GLfloat kBottom = -7.5f;
static const GLfloat kTop = 7.5f;
static const GLfloat kNear = 10.f;
static const GLfloat kFar = 100.f;
```

This results in a horizontal view angle of a 120° degrees with the near plane at z= –10, and the far plane at z= –100, see Figure 3.

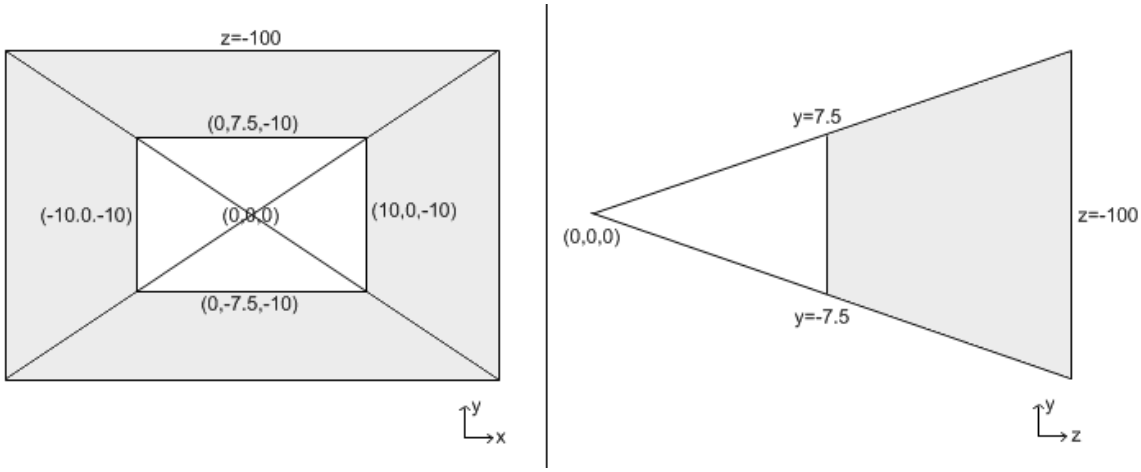

Figure 3. Frustum

The resulting area (colored in gray) is where the models have to be rendered to be visible to the user. The 120° degree viewing angle is obtained with the following calculations:

```
screen resolution = 1024 x 768
screen aspect = sAspect = 1024/768 = 4/3 = 1.333
d = absolute horizontal distance
near = desired near Z plane distance
horizontal viewing angle =2*asin(d/(2*near))
height=2((width/2)/sAspect)
```

Function parameters — `glFrustum(-width/2, width/2, -height/2, height/2, near, far)`

The horizontal viewing angle formula is obtained by getting a horizontal slice of the frustum and bisecting it, therefore obtaining two triangles with a 90° degree angle, see Figure 4.

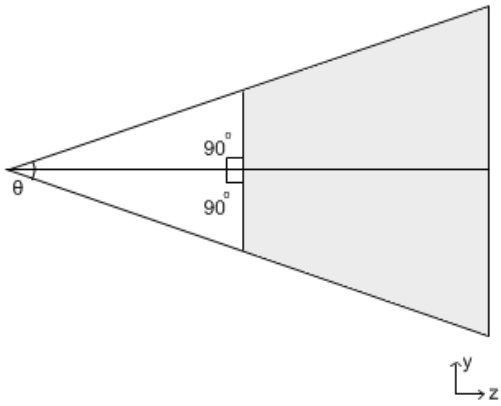

Figure 4. Bi-Section of Angles

If an angle of 120° degrees is desired use the below calculation:

```
120=2*asin(d/(2*near))
60=asin(d/(2*near))
1.732=d/near
```

A 1.732 ratio between the absolute horizontal distance and the nearest plane has to be respected. When calculating frustum dimensions keep in mind that both width and height are highly dependent on the screen aspect.

For more information on OGLES, please refer to Section 6, “References”.

Appendix C
Configuring DIU Kernel Modules

C.1 LTIB Configure Kernel

The DIU must be pre-configured when using a new 20080528 install. If there are any DIU problems, (for example not getting any video out of the DVI port) checking the correct DIU driver configuration in the kernel is a good place to start.

Run LTIB in the configuration mode, check Configure the kernel, exit, and save when prompted:

Make sure that the following modules in the Kernel configure tree view are built into the kernel:

```
Device Drivers --->
  Graphics Support --->
    Display device support --->
      <*> Display panel/monitor support
    <*> Support for frame buffer devices
    <*> Freescale MPC8610/MPC5121 DIU framebuffer support
    Console display driver support --->
      [*] VGA text console
      [*]  Enable Scrollback Buffer in System RAM
      (64)   Scrollback Buffer Size (in KB)
        <*> Framebuffer Console support
        [*] Select compiled-in fonts
        [*] VGA 8x8 font
        [*] VGA 8x16 font
        [*] Pearl (old m68k) console 8x8 font
        [*] Acorn console 8x8 font
        [*] Mini 4x6 font
        [*] Sparc console 8x16 font
        [*] Bootup logo --->
        [*] Standard black and white Linux logo
        [*] Standard 16-color Linux logo
        [*] Standard 224-color Linux logo
```

Save and exit the LTIB configuration screen, then run the install script. There should now be DIU support.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

AN3793
Rev. 0
12/2008