White Paper

3D Graphics Capabilities and Architecture

Broadcom provides 3D graphics capabilities in hardware and software through the OpenGL® ES API. This document describes these 3D capabilities and the application architecture used to support 3D in applications and middleware.

OpenGL ES (for Embedded Systems) is a low-level, lightweight API for advanced embedded graphics. Broadcom provides an implementation of the OpenGL ES API that offloads the 3D rendering to the 3D core. This allows set-top box applications to provide 3D effects with minimal impact on CPU performance.

January 2008

Table of Contents

OpenGL® ES 1.0

OpenGL ES (for Embedded Systems) is a low-level, lightweight API for advanced embedded graphics using well-defined subset profiles of OpenGL

There are many benefits to using an open standard, such as conformance tests, literature, applications and sample codes, specifications, and message boards.

The Broadcom OpenGL ES 1.0 driver provides a method for lighting and transforming 3D models into a world space and projecting them onto a 2D surface with clipping. Transformation, lighting, and some clipping is done using the CPU and FPU, and rendering and guard-band clipping is done by the 3D device.

Background

On July 28th, 2003, the Khronos group released an official subset of the OpenGL standard called "OpenGL ES" (http://www.khronos.org/opengles/), describing it as:

OpenGL® ES is a low-level, lightweight API for advanced embedded graphics using well-defined subset profiles of OpenGL. It provides a low-level applications programming interface (API) between software applications and hardware or software graphics engines.

This standard 3D graphics API for embedded systems makes it easy and affordable to offer a variety of advanced 3D graphics and games across all major mobile and embedded platforms. Since OpenGL ES (OpenGL for Embedded Systems) is based on OpenGL, no new technologies are needed. This ensures synergy with, and a migration path to, full OpenGL, the most widely adopted cross-platform graphics API.

The advantages of using Open GL ES include:

- Many customers are adopters of this standard. (http://www.khronos.org/members/conformant/).
- For adopters, there is a set of detailed conformance tests that may be used to verify both the software implementation and 3D hardware.
- There are code samples, specifications, and message boards to aid the support of developers, in addition to Broadcom support. (http://www.khronos.org/developers/).
- Books specifically written about OpenGL ES are available for purchase (OpenGL ES Game Development).
- External applications that use the subset of OpenGL defined here are available.

Broadcom now supports the OpenGL ES API as its official 3D graphics API. At the present time, the 3D core exists in the BCM7038 and BCM7400 devices. In the future, the 3D core may be included in a variety of Broadcom set-top devices.

Common Operations

There is a set of frequently used 3D operations that create an onscreen display on a set-top box. These are all easily and efficiently supported by the Broadcom 3D hardware and software. In some cases, the operation can be performed with the 2D core (such as scaling and moving in 2D space).

Transformation Operations

- **Move**: Objects can be moved in both 2D and 3D space. Moving an object in 2D space results in movement around the screen, but movement in 3D space will move the object closer and further from the screen. Only 3D renders will recognize movement in 3D space.
- **Scale**: Objects can be scaled independently in all three dimensions, although 2D objects can only be scaled in 2D.
- **Rotate**: Objects can be independently rotated around all three axes.

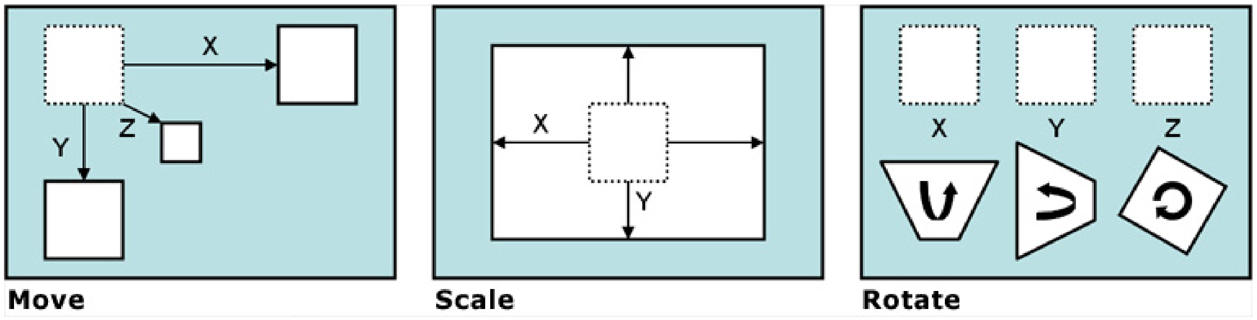

Figure 1: Transformation Operations

Pixel Operations

- **Color**: Objects can be colored with an image using pixel modulation.
- **Fade**: Objects can be faded to what is behind them using alpha blending.
- **Merge**: Objects can merge images together using alpha blending.

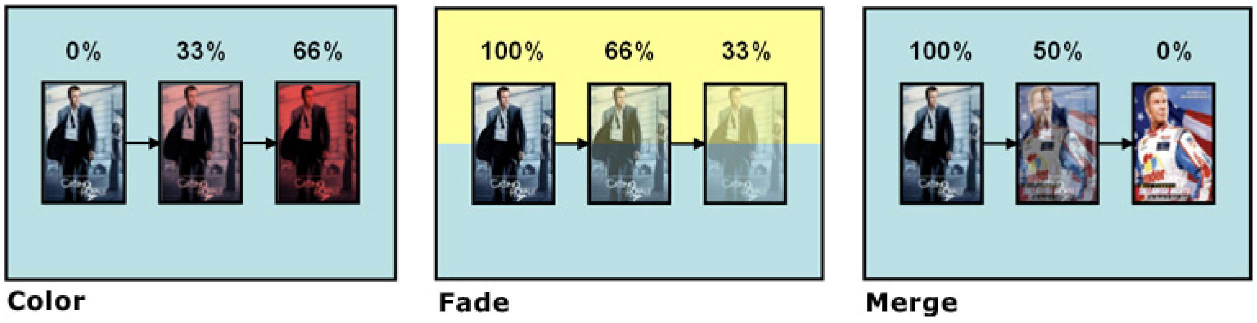

Figure 2: Pixel Operations

Polygon Operations

- **Reflection**: Objects can have a reflection underneath them. Only 3D renders will recognize a reflection due to the required gradient blend.

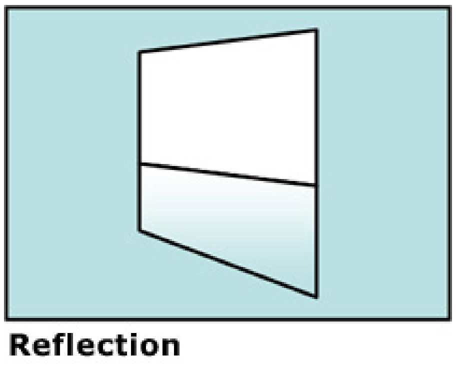

Figure 3: Polygon Operations

Architecture

Hardware Blocks

The hardware core that provides 3D acceleration, the PX3D, is independent and distinct from the core that provides 2D acceleration (the M2MC or Memory-to-Memory Compositor). These individual cores are described in this section.

The PX3D Device

The PX3D is the existing 3D core currently in use in the BCM7038 and BCM7400. It is an OpenGL ES 1.0-conformant graphics engine. This is a retained-scene, deferred-render system, capable of supporting the geometry and pixel rates of lighter 3D applications.

The PX3D is capable of processing between 100-300K polygons per second and can achieve a pixel rate of 20-40 Mpixels textured (32 bpp vs. 16 bpp), up to 100 Mpixels non-textured. It can support a scene complexity of 2K polygons with a limit of 64 triangles crossing any one scan line. It is a compact design at only 1.2 mm^2 in 65 nm.

The PX3D is a 3D scene-rendering device that renders all the polygons within a scene, scan by scan, once they have been submitted. A maximum of 64 polygons can cross any of the scans in the render. Compared with traditional triangle rendering, scene rendering has a number of benefits and special characteristics, described below:

Benefits:

- Full depth buffer functionality with no depth buffer.
- No depth buffer also means there is no need to read or write to one.
- Deferred texture mapping results in only reading texels that contribute to the final scene.
- Blending does not require reading previously rendered pixels.
- The entire scene is rendered in scan order, unless it is too complex.

Notes:

- Rendering does not start until all the polygons in the entire scene are submitted.
- Polygon data for the entire scene is stored in memory.
- Polygon positioning needs to be tracked and the scene should be split into tiles when it is too complex.

The PX3D device does not render as fast as the M2MC blits, although when polygons get overlaid and/or blended, there would be time saved due to deferred blending and texture mapping and the absence of a depth buffer.

Pixel Throughput

The PX3D device:

- Reads 40 million unrotated texels per second. Rotations around the Y-axis and, to a larger extent. the Z-axis cause texture cache misses, with larger rotations causing more misses.
- Writes 100 million pixels per second. Untextured solid color and gradient polygons render at this speed.

The Memory-to-Memory Compositor

The M2MC (Memory-to-Memory Compositor), or "blitter," is capable of combining rectangular source regions with a destination area, while performing a range of functions, including resize and logic ops. It is typically used for compositing regions on the OSD or graphics plane and scaling them to output resolutions.

The latest version of the M2MC in the BCM7405 operates at 216 MHz, processing two pixels per clock to give a blit rate of 432 Mpixels.

Usage with the Magnum Porting Interface

The OpenGL ES driver can be used in application that is written to the Magnum Porting Interface (PI) layer. In this usage, the application is responsible for providing a frame buffer to the driver. This frame buffer is used by OpenGL as the destination of the render, and it must be managed by the application so that the update is shown on screen.

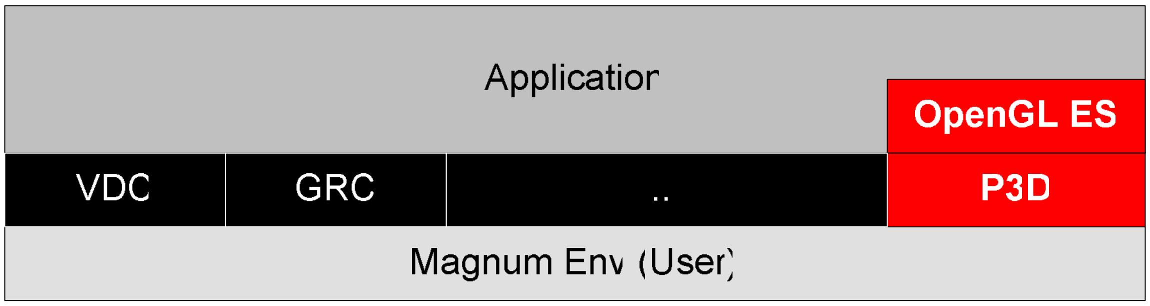

Figure 4: OpenGL ES and the

Magnum Porting Interface Layer

Usage with the the Settop API

The OpenGL ES driver can be used in application that is written on top of the Settop API. In this usage the OpenGL ES driver requests a frame buffer from the Settop API. The Settop API manages this buffer and updates the display as appropriate.

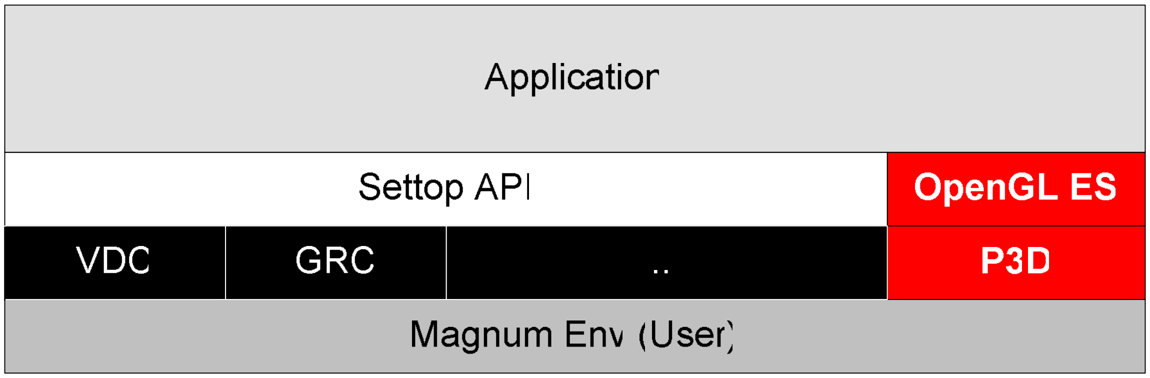

Figure 5: OpenGL ES and the Settop API

Usage with the Settop API and Linux Kernel Mode

The Porting Interface module that controls the 3D core (PX3D) cannot reside in Linux kernel space due to extensive use of floating point processing and the need maximize OpenGL ES efficiency. This means the PX3D PI module must be run in user space, regardless of where other Magnum components are executing.

At this time the PX3D PI requires user space versions of the following Magnum base modules: BCHP, BERR, BSTD, BDBG, BINT, BMEM, BKNI, and BREG.

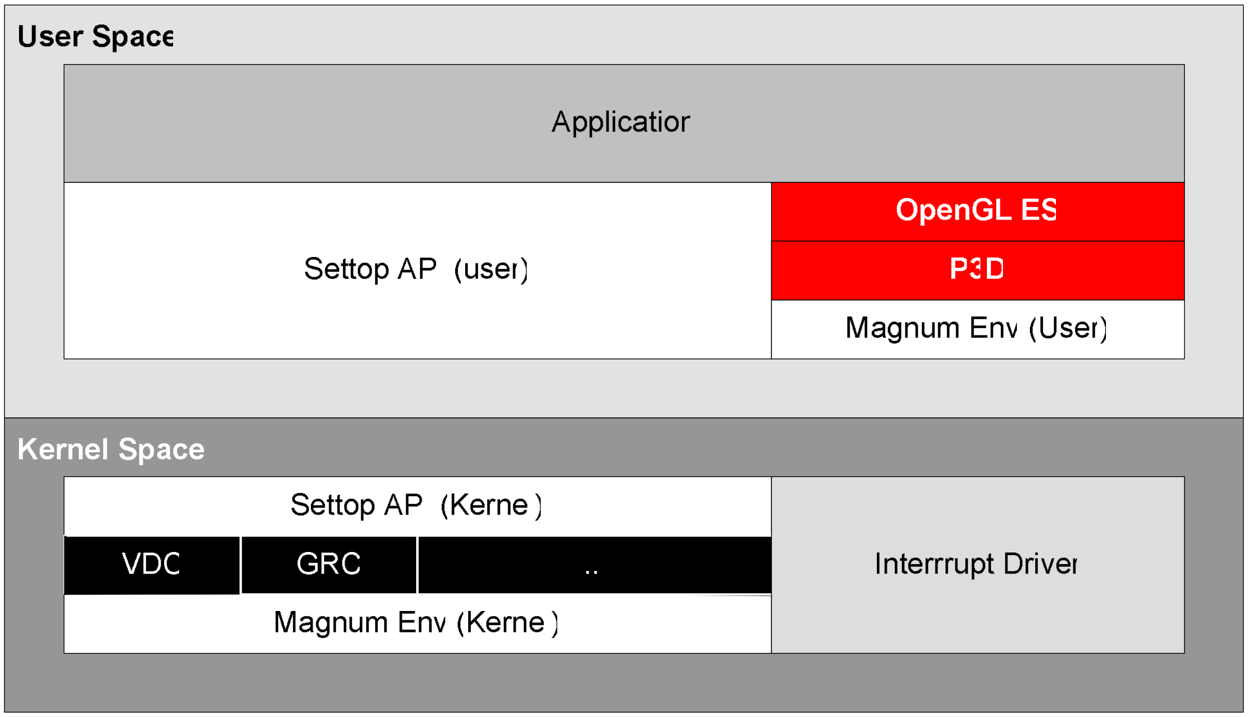

Figure 6: Linux Kernel Mode

These components are implemented and used when running the Broadcom reference software in user mode. However, the following changes must be made to the default user mode implementation of the following modules when reused specifically for the PX3D:

- BINT and interrupt driver: Only supports the 3D interrupt.
- BMEM: Heap size constrained to meet 3D requirements.

Usage Recommendations

Display Organization

Since 2D blits are faster than 3D rendering, the area rendered should be limited to ensure a fast frame rate. The 3D device can be expected to render between 10 to 40 millions texels per second, with the range depending mostly on scaling and rotation of textures. If we assume that there is not too much downscaling or Y- and/or Z-axis rotation, then we can expect about 25 to 30 million texels per second. At 60 frames per second, we can expect to render about 0.5 Mpixels per frame or 1 Mpixels at 30 fps. There are several ways to configure the display to reduce the amount of area that is being rendered to.

720x480

The 3D device can render full screen at 60 fps when the display is 720x480, unless there is excessive rotation, downscaling, or overlaying of transparent polygons. At 720x480, there are 346K pixels to render, which is 20 Mpixels/sec at 60 fps.

720 x 480
346K pixels

Figure 7: 720x480 Display Option

1280x720

The 3D device can only render full screen at 60 fps when the display is 1280x720 if much of the scene is untextured; otherwise, it can do it at 30 fps. An alternative to rendering at full screen is to render at half the horizontal size (640x720) and scale to the full display size with filtering using the graphics feeder. At 640x720, there are 461K pixels to render, which is 28 Mpixels/sec at 60 fps.

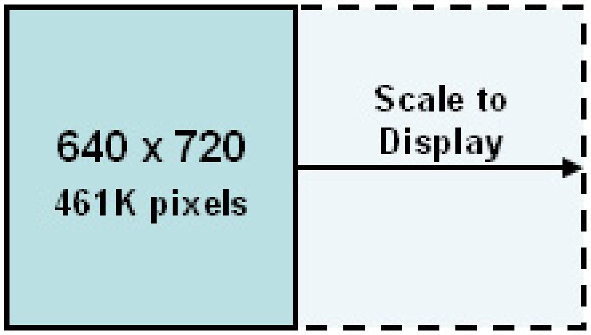

Figure 8: 1280x720 Display Option 1

Another alternative that provides higher resolution rendering is to render to only half the display, without any scaling. The 2D device would be used to draw to the rest of the frame using blits. This option is the most viable for achieving a high-resolution image at a high frame rate.

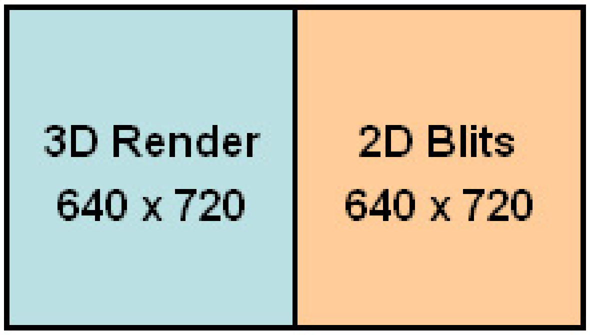

Figure 9: 1280x720 Display Option 2

1920x1080

The 3D device cannot render at full screen at 60 fps when the display is 1920x1080, but can only do it at about 15 fps. Instead, the 3D device could render to a quarter the display size (960x540), and have the 2D device scale it vertically to 960x1080 with filtering. The graphics feeder would then scale it horizontally to the display at 1920x1080, also with filtering. The filtering would also provide full scene anti-aliasing. At 960x540, there are 518K pixels to render, which is 31 Mpixels/sec at 60 fps.

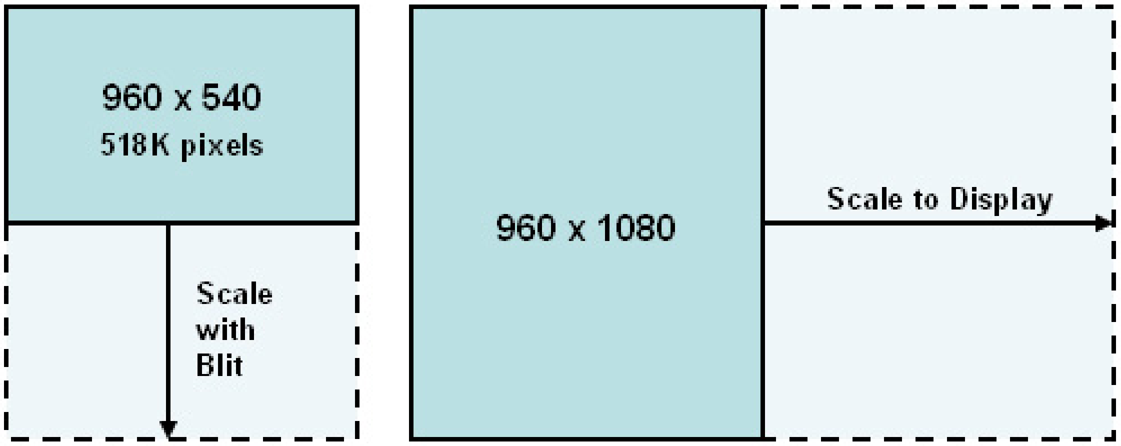

Figure 10: 1920x1080 Display

Rendering to only a quarter of the full display without any scaling is another option. This would allow for high quality rendering, but limit the amount of area that displays 3D graphics.

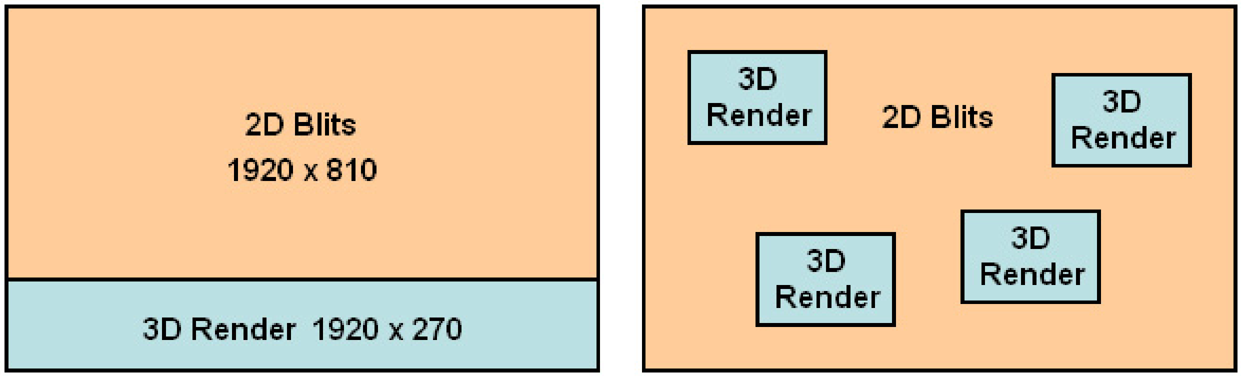

Figure 11: 1920x1080 Quarter Display

Open GL Usage

There are a few things to keep in mind when using OpenGL ES with the scene-rendering device:

- Fully clear the render target unless polygons are being blended with its contents. This will prevent the 3D device from reading the render target's memory.
- Fully clear the depth buffer even though it may not exist. The 3D device can render with or without a depth buffer, but clearing it indicates that its previous contents are not relevant to the current render.
- Use mipmap textures when possible. The 3D device selects the texture with a size that will map one of its texels to one pixel on screen. This prevents too much downscaling, which causes unnecessary texel reads.
- Do not modify textures that have been recently used in a render, since rendering is delayed until after all the polygons in a scene have been submitted.
- Clip to the bounds of the render. Since rendering will occur within a rectangle area, clipping will eliminate unnecessary pixel reads and/or writes.

Optimizations

Optimized Z-Buffer Management

The on-chip 3D render engine uses a proprietary Z-buffer calculation and ordering algorithm that avoid unnecessary memory writes and reads. No memory-instantiated Z-buffer is required for full Z-buffer functionality, although one can still be used to composite two separate frames/scenes if required. By default, no Z-buffer is allocated by the driver.

Clearing the entire render surface and the Z-buffer is recommended because it lets the driver know that the 3D render is not attempting to blend with what is already in the render surface or to use an uncleared Z-buffer.

Texture Management

Once textures have been loaded, do not subsequently modify the textures because they may not have been used in a previous render yet, which results in them getting copied.

Polygon Transformation Management

The OpenGL ES transformation pipeline uses a software-implemented routine leveraging the on-chip hardware floating point unit (FPU). The programmer should confirm that the complier environment selects the use of the FPU and not an alternative FPU emulation stack.

Together with the BCM7400 dual MIPS processor, the FPU-enabled transformation routine has sufficient processing power for about 2000 polygons per frame at 60 frames per second. Hence it is recommended that the polygon count per scene be kept to 2000 or less.

Rendering Optimization

Avoid creating and drawing into a full HD drawing surface. Instead, draw into a smaller surface and use the on-chip scaling engines to bring the graphic size to full screen. See the *Displaying 3D Graphics* section.

Do not read or write to a 3D scene before it has been completely displayed. Reading or writing to a 3D scene after drawing triangles to it and with the intent of drawing more triangles to it is supported, but will force the graphics engine to prematurely render the scene.

Use mipmapping, which reduces the size of a texture as an object moves further away resulting in reading less texels per polygon.

Optimized Text Drawing

When possible, use blits for drawing 2D objects, like text. Blits are much faster than the 3D renders and can, therefore, be done at full display resolution for better quality.

Integration

Open GL ES has been integrated into many products and middlewares applications in the embedded markets that require high-performance, low-power 3D graphics, including mobile (iPhone), gaming (Sony PS3), and STB markets.

This section contains a brief discussion of how OpenGL ES can be utilized in OCAP and Java® environments.

Java Binding–JSR 239

JSR 239 defines an optional package that may be run on a number of Java Micro Edition (Java ME) platforms with at least the capabilities of CLDC 1.1/MIDP 2.0 or CDC 1.1.2/PBP 1.1.2. OpenGL® ES is only supported for devices with displays.

The OpenGL ES is defined by the Khronos Group (http://www.khronos.org). OpenGL ES defines two profiles: the Common profile and the Common-Lite profile. The Common-Lite profile is a 32-bit fixed-point profile, while the Common profile supports floating point. The Common profile is a superset of the Common-Lite profile.

JSR 239 requires an underlying native engine that has been certified by Khronos to be conformant with the OpenGL ES API. This engine must support version 1.0 of the OpenGL ES and all core extensions associated with the API. It may optionally support version 1.1 of the OpenGL ES API. Bindings are also provided for all currently defined optional profile extensions, including the extensions in the OpenGL ES 1.1 Extension Pack. Extension methods that are not supported by the run-time OpenGL ES engine will throw an exception named java.lang.UnsupportedOperationException.

The JSR 239 reference implementation can be licensed from Sun Microsystems®.

JSR 239 Features

- Immediate mode API
  - Highly flexible control of rendering
- Low-level hardware-oriented 3D API
  - Potentially higher performance than SW-based technologies
- Key features
  - Full-featured 3D API
  - 3D-viewing pipeline
  - Lighting and shading
  - Texture mapping and cube maps
  - Fog
  - 2D sprites
  - Extensibility

OCAP

OCAP (like MHP) uses the HAVi extensions to AWT in order to meet the 2D graphics requirements posed by TV displays and the set-top box hardware architecture.

OpenGL ES extensions are handled in a similar way to extend the TV experience with 3D capabilities. The following diagram shows that JMF, AWT, HAVi, and OpenGL are at the same level in the OCAP architecture.

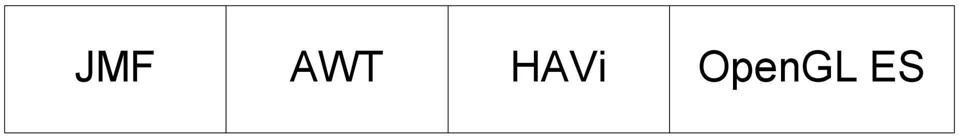

Figure 12: OpenGL ES in OCAP Architecture

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com
STB_3DGraphx-WP00-R

BROADCOM CORPORATION
5300 California Avenue
Irvine, California 92617
© 2008 by BROADCOM CORPORATION. All rights reserved.

01/16/08