**Freescale Semiconductor**
Application Note

Document Number: AN4132
Rev. 0, 05/2010

# 3D Math Overview and 3D Graphics Foundations

*by* *Multimedia Applications Division*
*Freescale Semiconductor, Inc.*
*Austin, TX*

This application note describes the basics of 3D graphics from basic terminology to specific i.MX MBX tips and tricks. Therefore, the user can understand and use the MBX graphics acceleration module.

**Contents**



## 1 Introduction

3D graphics is evolving, and most of the multimedia devices use real-time 3D graphics. This application note describes some conditions that the developers should understand to make real-time hardware accelerated 3D graphics work for games, Graphical User Interfaces (GUIs), 3D navigation devices, and so on.

## 2 3D Graphics and Real Time

3D graphics is widely used in many industries such as aerospace, medical visualization, simulation and training, science and research, and entertainment. 3D computer graphics uses the mathematical models (for example, groups of triangles or points) to represent a 3D object on the screen. The final image is a 2D image computed from various parameters such as position with respect to the viewer, lighting effects, and surface color.



freescale semiconductor

The process of making a 2D image from the 3D information is called rendering. The frame is sent to the display after it is rendered by the software and hardware. This process is repeated until the user halts it. The final displayable image is called a frame. Due to the nature of this process, the time taken for rendering is very small (typically 1/30th of a second).

The frame rate is the measure of the number of full screens (frames) that a given application refreshes or redraws per second. If the 3D graphics are rendered and displayed fast enough so that the user can interact with them, then it is called real time.

## 2.1 Software Rendering vs. Hardware Accelerated Rendering

There are two main ways to render 3D graphics:

- Software rendering
- Hardware accelerated rendering

### 2.1.1 Software Rendering

In software rendering, the rendering code runs on a general purpose Central Processing Unit (CPU) by using specialized graphics algorithms. This rendering is extremely slow because the scene complexity and frame resolutions are high. Software rendering is extensively used in the film industry to render frames.

### 2.1.2 Hardware Accelerated Rendering

As 3D graphics requires several computations for a stand alone CPU to handle the data in real time, a specialized real-time 3D graphics hardware has been developed. This is used in PCs, game consoles, and the latest embedded devices such as i.MX MBX technology.

# 3 MBX Module Overview

The MBX R-S 3D Graphics Core is an Advanced Microcontroller Bus Architecture (AMBA) compliant System-on-Chip (SoC) component. Figure 1 shows a top-level block diagram of the MBX R-S 3D Graphics Core.

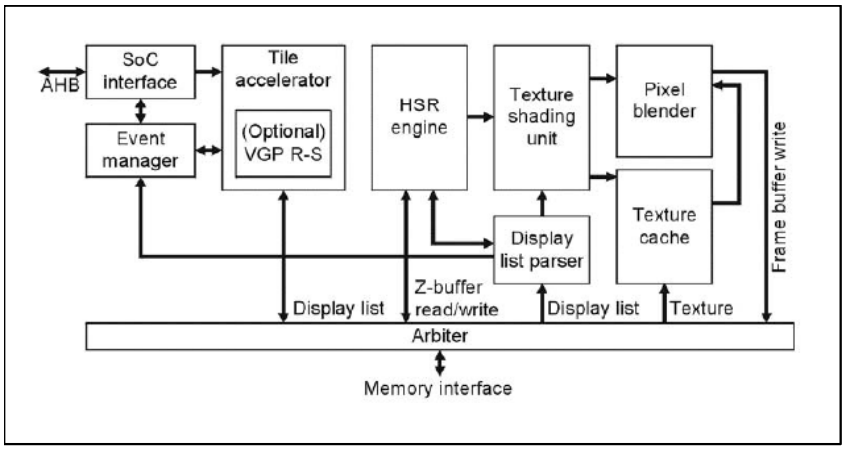


**Figure 1. MBX R-S 3D Graphics Core**

The MBX R-S 3D Graphics Core consists of the following modules:

- Tile Accelerator (TA)
- Event manager
- Display list parser
- Hidden Surface Removal (HSR) engine
- Texture shading unit
- Texture cache
- Pixel blender

The MBX R-S 3D Graphics Core operates on 3D scene data (sent as batches of triangles) that are transformed and lit either by the CPU or by the optional VGP R-S. Triangles are written directly to the TA on a First In First Out (FIFO) basis, so that the CPU is not stalled. The TA performs advanced culling on triangle data by writing the tiled non-culled triangles to the external memory.

The HSR engine reads the tiled data and implements per-pixel HSR with full Z-accuracy. The resulting visible pixels are textured and shaded in Internal True Color (ITC) before rendering the final image for display.

## 3.1 MBX R-S 3D Graphics Core Features

The MBX R-S 3D Graphics Core has the following features:

- Deferred texturing
- Screen tiling
- Flat and Gouraud shading

- Perspective correct texturing
- Specular highlights
- Floating-point Z-buffer
- 32-bit ARGB internal rendering and layer buffering
- Full tile blend buffer
- Z-load and store mode
- Per-vertex fog
- 16-bit RGB textures, 1555, 565, 4444, 8332, 88
- 32-bit RGB textures, 8888
- YUV 422 textures
- PVR-TC compressed textures
- 1-bit textures for text acceleration
- Point, bilinear, trilinear, and anisotropic filtering
- Full range of OpenGL and Direct3D (D3D) blend modes
- Dot3 bump mapping
- Alpha test
- Zero cost full scene anti-aliasing
- 2D via 3D

**NOTE**

The MBX module is present in the i.MX31 processor, but not in the i.MX31L processor.

# 4 3D Graphics in a Nutshell

Rendering hardware is built primarily to draw 3D triangles. However, setting up and manipulating the 3D triangles involve algorithms that use 3D mathematics and other techniques.

## 4.1 Coordinate Systems

A coordinate is a series of numbers that describes the location in the given space. 3D graphics system operates in a mathematical space. The space used in most of the 3D graphics is called 3D Cartesian coordinate. The Cartesian coordinate system uses a series of intersecting line segments to describe a location with respect to the origin. The origin is a point in the space where all the coordinates are 0. The intersecting lines are orthogonal or perpendicular to each other.

Figure 2 shows the 3D Cartesian coordinate system.

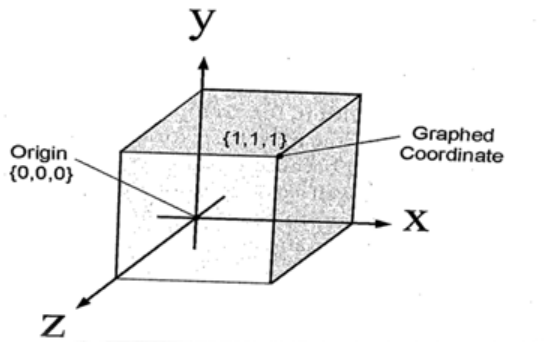


**Figure 2.** 3D Cartesian Coordinate System

The intersecting lines are named as X-axis, Y-axis, and Z-axis by convention. The standard order is a right-handed orientation.

## 4.2 3D Objects and Polygons

A 3D model is composed of relational and geometric information. This information is generally stored in the form of polygons and vertices. A polygon is a multi-sided closed surface that consists of vertices that are connected by chained lines. The coordinates of a polygon are stored in a vertex, and each vertex is associated with a color. A triangle, which has three vertices, is the most basic form of a polygon. It is planar and convex, which is essential for lighting and collision detection.

3D objects do not necessarily have to be made of only triangles. However, the objects are generally triangles or converted to triangles because they can be handled easily. 3D objects are composed of triangle meshes (arrays). This geometric data can be imagined as a set of coordinates or points that has a common origin, and the triangle sets are made with these coordinates.

Figure 3 shows a polygon.

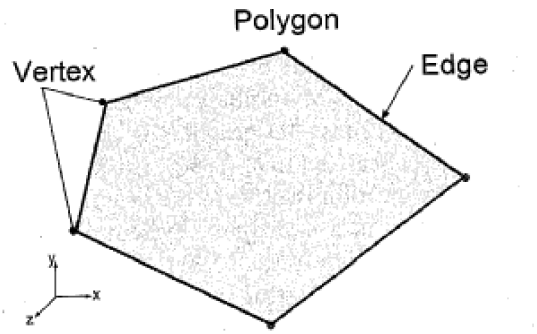


**Figure 3. Polygon**

Another important concept is the winding order. It determines the front and back of a polygon. The default winding order of a polygon in OpenGLExtractor (OpenGL ES) tool is counterclockwise. This order can be changed while rendering. However, the winding order is generally taken as counterclockwise.

Some 3D objects lend themselves to be generated with code such as terrain, unlike the video game characters. The video game characters are generated by the 3D modeling software. Sophisticated software applications are available to create 3D geometry for these cases, such as the popular 3D Studio Max and Maya.

## 4.3 Transformations

Transformation is an operation that uniformly changes the coordinates of a piece of geometry. Here, the given operation is performed on each vertex and the overall shape is preserved. Therefore, transformation creates a change in the 3D object or coordinate system.

3D transformations are generally stored as 3D matrices. However, the matrices are not processed directly by the code, but are abstracted by some form of a transformation class.

There are three major types of transformations in a 3D graphics system:

- Translation
- Rotation
- Scaling

### 4.3.1 Translation

In translation, all the points or vertices in the object are moved along a single axis (X,Y or Z).

Figure 4 shows translation of a cube with the translation matrix.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

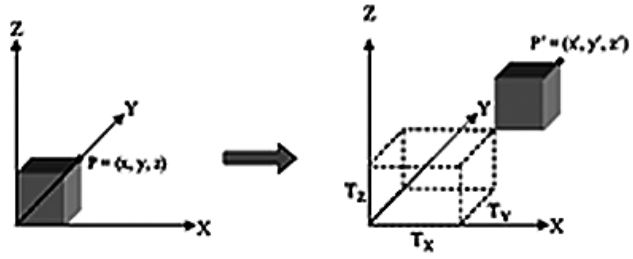


**Figure 4. Translation in 3D and Corresponding Translation Matrix**

### 4.3.2 Rotation

In 3D, rotation occurs about an axis. The standard way to rotate is by using the left-handed convention.

Figure 5 shows rotation about X and Z axes and their corresponding rotation matrices.

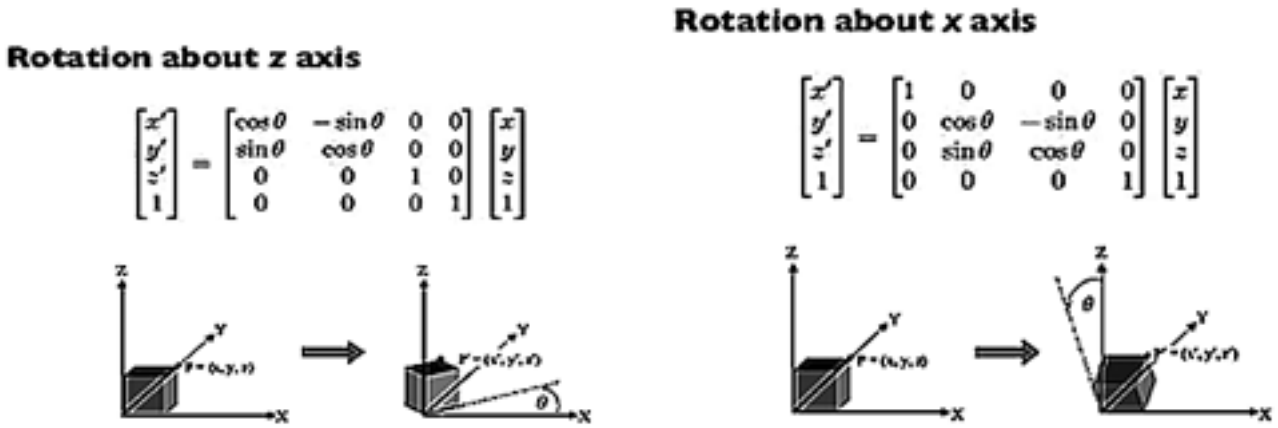


**Figure 5. Rotation in 3D with Corresponding Rotation Matrices**

### 4.3.3 Scaling

An object can be scaled to make it proportionally bigger or smaller by a factor of k. If the same scale is applied in all the directions, then the scaling is being performed in a uniform scale. Thus, the object is dilated about the origin.

Uniform scaling preserves the angles and proportions. If all the lengths in uniform scaling increase or decrease by a factor of k, then the areas change by a factor of $k^2$ and the volumes (in 3D) by a factor of $k^3$.

Figure 6 shows a scaling matrix.

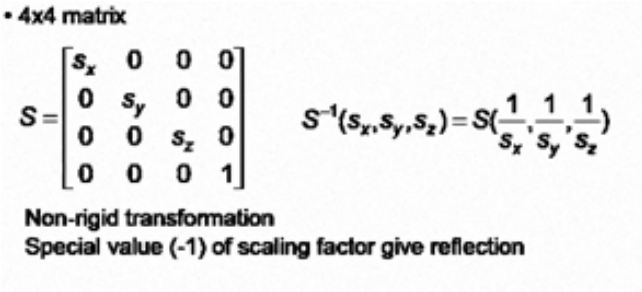


**Figure 6. Scaling Matrix**

## 4.4 Camera and Projection

The final render of a 3D object or scene is a 2D image. The process of obtaining a 2D display from 3D virtual space involves two important concepts: camera and projection.

### 4.4.1 Camera

The idea of camera is that the developer can place a camera virtually in the 3D world and have the system render from that point of view. However, a camera is an illusion in which the captured objects are inversely transformed by a camera transform. A real camera does not exist in an OpenGL tool. Alternatively, a

ModelView matrix is used to create the effect of a camera. Camera manipulation is important in 3D applications. The application should implement camera classes to make camera manipulation easier.

### 4.4.2 Projection

Projection is carried out on 3D polygons and is converted into 2D polygons by using the projection matrix. The projection matrix is created by the rendering system that is based on the state of the camera and other factors such as the field of view. In general, projections transform points in an N dimensional coordinate system into a coordinate system of dimension less than N. The 3D hardware handles this process automatically. Therefore, the initial setup alone is generally sufficient.

### 4.4.3 Camera Model

OpenGL draws primitives according to a user-definable camera model. A camera can be specified in intuitive terms: position, aperture, and so on. Several cameras can be used based on the purpose. For example, a mini-map in the corner of a screen can use different camera parameters as the main scene. Additionally, selection can be made from different projection schemes such as perspective and orthographic. Therefore, the desired viewing algorithm can be chosen. The user can also completely override the camera and draw in the window coordinates. This is useful while drawing in 2D, performing overlays for menus or interface elements, and so on.

An OpenGL camera is a series of transforms. The full scene is rotated according to the camera's orientation and then translated, so that the viewpoint is placed correctly. An optional perspective transform is then applied. Hand coding of these transforms is troublesome and prone to errors. Thus, the Application Program Interface (API) of the PowerVR Software Development Kit (PVR SDK) provides easy-to-use calls that handle the camera operation. Two calls are involved in the camera operation. The first call allows specification of the camera projection properties (aperture, aspect ratio, type of projection, and so on). The second is used to directly place the camera in the 3D world.

The minimum parameters that are required to set up a virtual camera are as follows:

- fFOV—Indicates the field of view, in degrees, in the Y direction.
- Aspect—Specifies the ratio between width and height. For example, a value of 1.3 implies that the width of the camera's snapshot is 1.3 times larger than the height.
- CAM_NEAR—Used to initialize the Z-buffer. This parameter is strictly a positive double value, specifying the distance (in Z) to the near clipping planes.
- CAM_FAR—Used to initialize the Z-buffer. This parameter is strictly a positive double value, specifying the distance (in Z) to the far clipping planes.

These parameters specify a view frustum completely as shown in Figure 7.

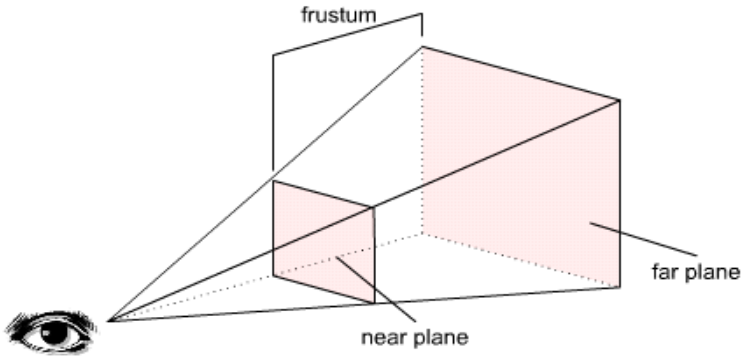


**Figure 7. View Frustum**

## 4.5 Lighting and Shading

Lighting is one of the most important elements in 3D graphics. Real light is made up of photons that form the fundamental particle of light. Trillions of photons interact in a simple lit surface. Light also has special behavior characteristics such as refraction, reflection, and obstruction that can further complicate the render. Therefore, accurate simulation of real light is a difficult process.

As the accurate simulation of light is computationally complex, lighting models have been developed to create the illusion of surface illumination on 3D models. These models are approximations of the real effect of light and can not be recognized by the human eyes.

### 4.5.1 Ray Tracing

Ray tracing, which is used in computer animation, employs the computer to create realistic graphic images. This process is done by calculating the paths taken by the light rays to hit the objects from various angles that creates shading, reflections, and shadows. These effects give the image a convincing look. However, ray tracing is computationally intensive and is inadequate for real-time rendering. Therefore, other models that are faster, but less accurate have been developed for real time.

Figure 8 shows the ray tracing process.

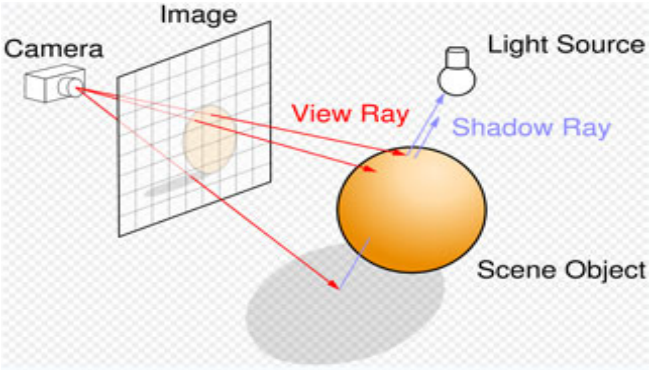


**Figure 8. Ray Tracing**

### 4.5.2 Real-Time Lighting and Shading

Real-time lighting can be categorized into two types:

- Static
- Dynamic

#### 4.5.2.1 Static

Static shading is performed when the shading effect is to be permanently colored into the 3D object and does not require the color to be changed during the run time. This technique is used on world models and relies on the 3D content-creation software for the shading effect.

#### 4.5.2.2 Dynamic

Dynamic shading is used on moving 3D objects and lights and is computed during the rendering of each frame. Dynamic shading uses 3D vector math and incorporates ambient illumination as well as diffuse and specular reflection of directional lighting.

The shading function is one of the simplest vector operations done in 3D graphics. As it is done many times in a typical scene, the hardware should be built to accelerate this computation. The basic shading function for the static or dynamic shading is a modulation of the polygon surface color, based on its angle to the light direction. These parameters are expressed as vectors and vector operations. Computing the shading function in real-time 3D graphics is a feature built into the 3D hardware and is not reimplemented by the developer. However, it is useful to understand the process, as it directly affects the procedure to make 3D models.

The steps for shading are as follows:

1. The dot product of the unit vector in the direction of light and the surface normal is computed. The resulting scalar value should be between –1.0 and 1.0.
2. The value is clamped in the range 0.0–1.0, and now the value represents the intensity of light for that surface. The surface color is multiplied or scaled with the intensity, and as the intensity is between 0.0 and 1.0, the color becomes darker accordingly, as shown in Figure 9.

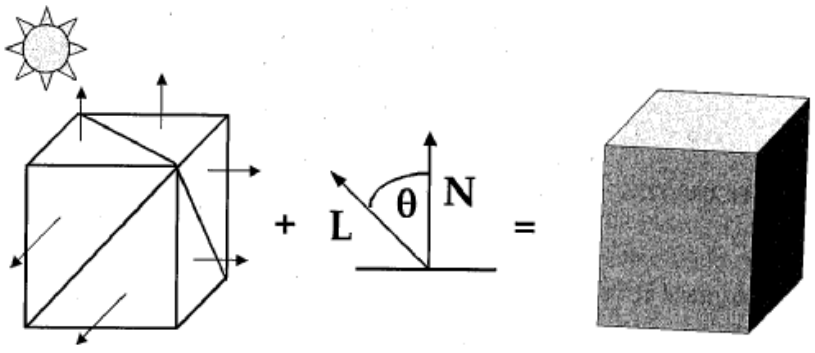


**Figure 9. Surface Color Scaled with Intensity**

3. The given triangle is then rendered with the resulting color for each surface.

### 4.5.3 Types of Shading

This section explains the different types of shading.

#### 4.5.3.1 Lambert or Flat Shading

Lambert or flat shading is the simplest among the different types of shading. Here, the shading function is calculated and applied for each polygon surface of the given object. Each polygon is uniformly colored based on the surface normal and direction of the light. Flat shading is a quick process; however, this process gives a faceted appearance to the object.

#### 4.5.3.2 Gouraud or Smooth Shading

Gouraud developed a technique to smoothly interpolate the illumination across the polygons and create a shading which is smooth and continuous. This shading technique is called Gouraud or smooth shading. This technique is effective and works on triangle-based objects, which are not actually smooth and continuous.

Figure 10 shows a comparison between flat and Gouraud shading.

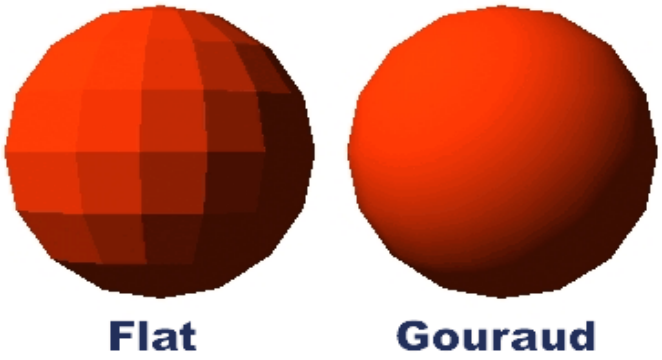


**Figure 10. Flat and Gouraud Shading**

A vertex normal is added to each vertex in a 3D object. This new normal at the vertices can be calculated by averaging the adjoining face normals. The vertex intensity is then calculated with that new normal and the shading function. The intensity is interpolated across the whole 3D mesh based on the vertices' attributes such as vertex color. OpenGL and modern 3D hardware support both flat and Gouraud shading models. Though both the shading techniques are used, smooth shading is more realistic.

### 4.5.4 Types of Lights

This section describes the different types of hardware-supported lights.

#### 4.5.4.1 Ambient or Omni Directional

Ambient light comes from several directions due to multiple reflections and emissions from several sources. The resulting surface illumination is uniform.

#### 4.5.4.2 Directional or Global

Directional light comes from a source located at the infinity. Therefore, directional light consists of parallel rays from the same direction. As the intensity of directional light does not diminish with distance, identically oriented objects of the same type are illuminated in the same way.

#### 4.5.4.3 Positional, Point or Local

Positional lights originate from a specific location. The light rays that emanate from the source are not parallel and can diminish in intensity with distance from the source. Identically oriented objects of the same type are illuminated differently, depending on the position with respect to the light source.

### 4.5.5 Shadow Problem

In real-time 3D graphics, as the lighting and shading models are local, shadows are not automatically rendered. Hardware-based real-time lights do not have complete built-in features such as shadowing and light occluding. The 3D object is lit regardless of the other objects blocking the light, as each triangle shading is computed independently of other triangles in the scene. To compute each triangle, considering all the scene geometry is impractical. However, the shadowing effects exist and are added in real time. There are several ways to create the effect of shadows. These ways are normally the combination of a 3D feature and a clever 3D engine. However, classic techniques include simple textured polygons, projected geometry, and more advanced techniques, such as using a hardware feature like the stencil buffer and the depth buffer.

# 5 Conclusion

3D math is an essential skill for a graphics programmer or real-time programmer. Knowledge of mathematical calculations behind the algorithms, SDKs, frameworks or engines helps to develop graphical or real-time programs. A solid knowledge in 3D math is necessary to explore the capabilities of the i.MX31 MBX graphics accelerator.

The information in this application note is a brief introduction to the world of mathematics behind real-time rendering and a good guide to mathematical concepts in the 3D program development. For a better understanding of 3D math, refer to Section 6, “References.”

# 6 References

The references for the application note are as follows:


- *3D Math Primer for Graphics and Games* by Fletcher Dunn & Ian Parberry, 2002 Wordware Publishing
- *Real Time Rendering* by Tomas Akenine/Eric Haines, Second Edition, 2002 published by AK Peters

# 7 Revision History

Table 1 provides a revision history for this application note.

**Table 1. Document Revision History**

| Rev. Number | Date | Substantive Change(s) |
|---|---|---|
| 0 | 05/2010 | Initial Release |

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

***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
1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com



Document Number: AN4132
Rev. 0
05/2010