



Tutorl3D

User Guide

80-V6449-1 Rev. A

August 22, 2003

**Submit technical questions at:
<https://brew-support@qualcomm.com>**

QUALCOMM Proprietary

Restricted Distribution: This document contains critical information about QUALCOMM products and may not be distributed to anyone that is not an employee of QUALCOMM without the approval of Configuration Management.

All data and information contained in or disclosed by this document is confidential and proprietary information of QUALCOMM Incorporated, and all rights therein are expressly reserved. By accepting this material the recipient agrees that this material and the information contained therein is held in confidence and in trust and will not be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of QUALCOMM Incorporated.

QUALCOMM is a registered trademark and registered service mark of QUALCOMM Incorporated. cdma2000® is a registered trademark of the Telecommunications Industry Association (TIA USA). Other product and brand names may be trademarks or registered trademarks of their respective owners.

Export of this technology may be controlled by the United States Government. Diversion contrary to U.S. law prohibited.

QUALCOMM Incorporated
5775 Morehouse Dr.
San Diego, CA 92121-1714
U.S.A.

Contents

| | |
|--|-----------|
| 1 Introduction..... | 4 |
| 1.1 Purpose | 4 |
| 1.2 Scope | 4 |
| 1.3 Conventions..... | 4 |
| 1.4 Revision history..... | 5 |
| 1.5 Technical assistance | 5 |
| 1.6 Acronyms | 5 |
| 2 Tutorl3D Application | 6 |
| 2.1 Starting-up | 6 |
| 2.2 UI notes | 7 |
| 2.3 Application event processing | 9 |
| 3 Tutorl3D Tutorials..... | 10 |
| 3.1 Transformations..... | 10 |
| 3.1.1 Rotation | 10 |
| 3.1.2 Translation | 12 |
| 3.2 Projections | 13 |
| 3.2.1 Focal length | 14 |
| 3.2.2 View depth..... | 15 |
| 3.2.3 Screen mapping | 17 |
| 3.2.4 Clipping rectangle..... | 19 |
| 3.3 Lighting and materials..... | 21 |
| 3.3.1 Direction and color | 21 |
| 3.3.2 Material..... | 24 |
| 3.4 Textures and blending | 27 |
| 3.4.1 Texture rendering | 27 |
| 3.4.2 Alpha blending | 31 |
| 3.4.3 Perspective correction..... | 33 |

Figures

| | |
|--|----|
| Figure 2–1: Keypad layout and key names | 7 |
| Figure 2–2: TutorI3D Main Menu | 8 |
| Figure 2–3: (a) Projections screen (b) View Depth tutorial screen | 8 |
| Figure 3–1 Rotation tutorial screen shot | 10 |
| Figure 3–2: Translation tutorial screen shot | 12 |
| Figure 3–3 Focal length tutorial screen..... | 14 |
| Figure 3–4 View depth tutorial screen shot | 15 |
| Figure 3–5 Screen mapping tutorial screen shot..... | 17 |
| Figure 3–6 Clipping rectangle tutorial screen shot | 19 |
| Figure 3–7 Lighting direction and color tutorial screen shots showing different lighting modes..... | 22 |
| Figure 3–8 Material tutorial screen shots. (a) shiny green material (b) dull green material..... | 25 |
| Figure 3–9 3D object rendered with (a) flat shading and (b) smooth shading..... | 28 |
| Figure 3–10 Texture rendering tutorial screen shot..... | 29 |
| Figure 3–11 Screenshots of the alpha blending tutorial showing various levels of alpha..... | 31 |
| Figure 3–12 Screen shots of the perspective correction tutorial showing perspective correction (a) enabled and (b) disabled | 33 |

Tables

| | |
|---|----|
| Table 1-1 Revision history | 5 |
| Table 3-1 Command keys for the Rotation tutorial | 11 |
| Table 3-2 Command keys for the Translation tutorial | 12 |
| Table 3-3 Command keys for the focal length tutorial | 14 |
| Table 3-4 Command keys for the view depth tutorial | 16 |
| Table 3-5 Command keys for the screen mapping tutorial..... | 18 |
| Table 3-6 Command keys for the clipping rectangle tutorial | 19 |
| Table 3-7 I3D lighting modes and properties | 21 |
| Table 3-8 Command keys for the lighting direction and color tutorial..... | 23 |
| Table 3-9 Command keys for the material tutorial | 26 |
| Table 3-10 I3D texture wrap modes | 27 |
| Table 3-11 I3D render modes | 28 |
| Table 3-12 Command keys for the texture rendering tutorial..... | 29 |
| Table 3-13 Command keys for the alpha blending tutorial..... | 32 |
| Table 3-14 Alpha value levels, ranges, and number of values in range..... | 32 |

1 Introduction

1.1 Purpose

The TutorI3D application is an interactive tutorial of the I3D Graphics API. Its purpose is to demonstrate the capabilities of I3D by allowing users to interactively change parameters and states, and visualize the resulting changes. Also, TutorI3D serves as an example of a dynamically downloadable I3D BREW™ application, and as such is a good starting point for application developers.

The purpose of this document is to give a general overview of the application, including the user interface, and to explain certain aspects of the source code, particularly the 3D graphics-related code. The mechanisms required to create a dynamic I3D application for the BREW platform are also discussed.

The majority of I3D APIs are demonstrated in the application. By reading this document and looking at the application, one will gain a good understanding of I3D and how to incorporate 3D graphics features, such as models, transformations, and lighting into an I3D BREW application.

1.2 Scope

This document is meant to complement the TutorI3D application. As such, this document is aimed at anyone who is interested in familiarizing themselves with the I3D API. All the information needed to develop an I3D application on the BREW platform is presented. This document assumes the reader understands the basics of 3D graphics terms and techniques, and has a reasonable understanding of the BREW API. The I3D API guide should accompany this document as references are made to API functions throughout this document.

1.3 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font. For example: `#include`

Code variables appear in angle brackets. For example: `<number>`

Commands and command variables appear in a different font. For example: **copy a:*. * b:**

Parameter types are indicated by arrows:

- Designates an input parameter
- ← Designates an output parameter
- ↔ Designates a parameter used for both input and output

1.4 Revision history

The revision history for this document is shown in Table 1-1.

Table 1-1 Revision history

| Version | Date | Description |
|---------|----------|-----------------|
| A | Aug 2003 | Initial release |

1.5 Technical assistance

For assistance or clarification on information in this guide, email QUALCOMM CDMA Technologies at **brew-support@qualcomm.com**.

1.6 Acronyms

For definitions of terms and abbreviations, refer to *Application Note: Software Glossary for Customers* (CL93-V3077-1).

2 TutorI3D Application

The TutorI3D application is an interactive tutorial on the I3D API. It demonstrates how to use BREW 3D APIs, I3D, I3Dmodel, and I3Dutil.

It contains tutorials which demonstrate the following I3D features:

- Transformations
- Projections
- Lighting and materials
- Textures and blending

2.1 Starting-up

You can run TutorI3D in the BREW Emulator on a PC or on a target phone that supports BREW. To run in the Emulator, you will need the BREW SDK installed on your system, as well as the I3D Extension. Both of these can be downloaded from the BREW website. Once the SDK is installed, check to see if the file named I3DExtension.dll exists in the Bin\Modules directory of the BREW SDK. If it does not, you will first need to install the extension using the instructions provided with the extension. Once this is done, and the TutorI3D is on your system, start-up the BREW emulator. At this point you will need to point the emulator to the directory where TutorI3D is installed. To do this:

1. Go to File —> Change Applet Dir, select the directory where the TutorI3D.mif file is installed, then click OK. This should be one directory above where the TutorI3D.dll is installed.
2. Next, go to Tools —> Settings, verify that the Specify MIF Directory check box is not checked, then click OK.

When using the left and right arrow keys, you should be able to scroll to the TutorI3D application icon.

3. Once it is selected, to start up the application click SELECT on the phone image (same as hitting ENTER on the keyboard). You should now see the Main Menu for the TutorI3D application.

To run on a target device, you will need a data connection between your computer and the device, and a tool to browse the device's file system. Using this tool, you must first create a directory named `tutori3d` (all lowercase), in the root brew directory on the device. Copy the `tutori3d.mif` file from your computer to the root brew directory on the device. Also, copy the following files to the newly created `tutori3d` directory:

- ❑ `tutori3d.mod`
- ❑ `tutori3d.bar`
- ❑ all files ending in a `.q3d` extension. These are the model files used by the app
- ❑ The signature file for the device. This should accompany the device

The required files are now on the device, and you can start-up TutorI3D as you would any other BREW application on the device.

2.2 UI notes

Throughout this document, key names will be used. Figure 2–1 provides a layout of a keypad with key names which may not be obvious. The displayed keypad is that for the toucan display. Note that your keypad may look quite different and the location of the keys may also be different. Consult your device's manual for the location where the CLR, SELECT, SEND and END keys are located.



Figure 2–1: Keypad layout and key names

The first screen you will see on the starting-up TutorI3D is the Main Menu as seen in Figure 2–2.

NOTE The fonts shown in the screen shots in this document may be different from the fonts you will actually see when running TutorI3D, depending on which BREW fonts are installed.

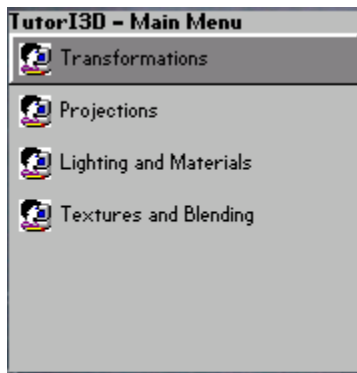


Figure 2–2: TutorI3D Main Menu

Using the up and down arrow keys, you can change selections on the main menu.

- Pressing the **SELECT** key will bring you forward one menu. For example, pressing **SELECT** on **PROJECTIONS** in the main menu will bring you to the screen shown in Figure 2–3a. At the bottom of this screen are the menu options for Projections. Use the left and right arrows to scroll between them, and press **SELECT** to enter into one of the tutorials. Figure 2–3b provides the screen for the View Depth tutorial.

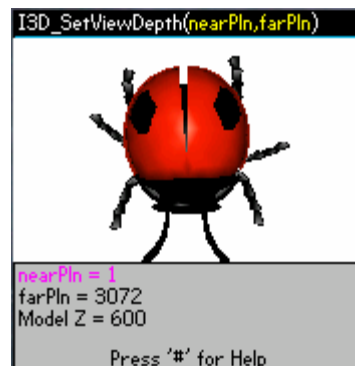
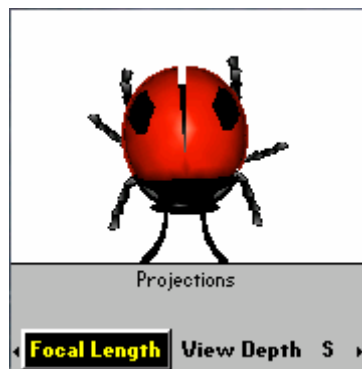


Figure 2–3: (a) Projections screen

(b) View Depth tutorial screen

The tutorial screen is broken up into three parts. At the top of the screen is the API function related to the tutorial. In Figure 2–3b, the function is `I3D_SetViewDepth()`. The parameters to the function are also specified, and the ones that are variable, meaning these the user can change dynamically in TutorI3D, are given in a yellow font color. The middle part of the screen is the 3D drawing window where the scene will be drawn. The bottom of the screen is the information area where the current values of the parameters are given, as well as other relevant information.

In any of the tutorial windows, you can press # to get a help listing for the current tutorial. This will tell you what keys do what in this tutorial. It is recommended that you take a look at the help menu every time you are entering a new tutorial to get an idea of the functional keys. Chapter 3 of this document explains the key commands for each tutorial in greater detail than the help menus built into the application.

- Pressing # in any of tutorials will bring up the help screen for that tutorial. Pressing it again, will hide it.
- Pressing CLR in any of the menus or tutorials will take you back one screen
- Pressing END at any screen exits the application

2.3 Application event processing

For application event processing:

- `EVT_APP_START` – initialize and bring up the main menu of the TutorI3D application
- `EVT_APP_STOP` – exits the TutorI3D, and frees all necessary data

3 TutorI3D Tutorials

3.1 Transformations

The transformation tutorials in TutorI3D are rotation and translation. You can enter the rotation tutorial, rotate a model, and then enter the translation tutorial to translate the model in its already rotated state. Or, vice-versa, you can first translate the model then proceed to rotate him in the already translated position. In other words, the two tutorials allow you to arbitrarily rotate and translate a model.

3.1.1 Rotation

The rotation tutorial allows you to arbitrarily rotate 3D models around the x, y, z axis using the I3D API. The main API function utilized to do this is `I3DUtil_GetRotateMatrix()`. A screen shot of this tutorial is provided in Figure 3–1.

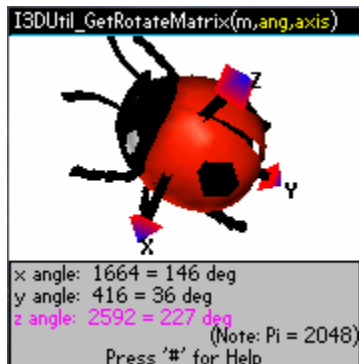


Figure 3–1 Rotation tutorial screen shot

This tutorial provides a model in the center and the model's axis extruding from the model. The bottom of the screen provides the amount of rotation in each axis in Q12 format, as well as in degrees. The currently selected rotation axis is shown in pink. Important to note is the fact that these rotations are based on the model's local axis.

Later in the translation tutorial, the model is translated along the screen x,y,z. Table 3-1 provides the command keys for this tutorial.

Table 3-1 Command keys for the Rotation tutorial

| Key | Action |
|------------|-------------------|
| Up arrow | Increase rotation |
| Down arrow | Decrease rotation |
| 1 | Select x axis |
| 2 | Select y axis |
| 3 | Select z axis |

At the top of Table 3-1, you can see that ang (angle of rotation), and axis (the axis to rotate about) are variables. Users select which axis they want to rotate about and increase/decrease the angle of rotation about that axis.

Note that the angle of rotation in Q12 is with respect to 2048 being PI. If you want to rotate by $\pi/4$ radians, the Q12 angle is $2048 / 4 = 512$, and if you want to rotate by x-degrees, the Q12 angle is $2048/180 * x$. The information area of the screen provides both the Q12 rotation angle, and the number of equivalent degrees for each axis.

Code example

The following is the example code for increasing the rotation of a model around the x-axis, and drawing it. The `I3D_StartFrame()` will need to be called to actually begin rendering the frame.

```

RotateXAndDraw( MyApp* pMe, AEE3DTransformMatrix* transform, int
rotation_angle)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // pMe->pModel: Pointer to the model instance
    // pMe->m_p3DUtil: I3DUtil instance created on application init
    // transform: current transformation matrix for the model
    // rotation_angle: angle to rotate in Q12

    AEE3DTransformMatrix m1;
    Do range checking on rotation_angle (range is -4096 to 4096, ie. -2PI to
    2PI)
    I3DUtil_GetRotateMatrix(pMe->m_p3DUtil, rotation_angle, &m1,
    AEE3D_ROTATE_X); I3DUtil_MatrixMultiply(pMe->m_p3DUtil, transform, &m1);
    I3DModel_SetSegmentMVT(pMe->pModel, transform, -1);
    I3DModel_Draw(pMe->pModel, pMe->m_p3D);
}

```

3.1.2 Translation

The translation tutorial allows you to translate a model arbitrarily along the x,y,z axis using the I3D API. The main API function utilized to do this is `I3DUtil_SetTranslationMatrix()`. A screen shot of this tutorial is shown in Figure 3–2.

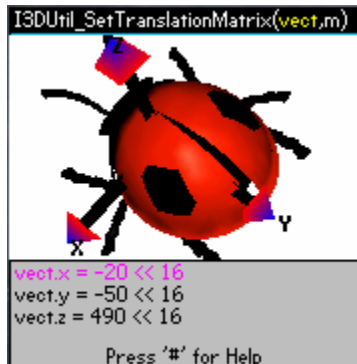


Figure 3–2: Translation tutorial screen shot

This tutorial provides a model and the model's axis extruding. The bottom of the screen provides the amount of translation in each axis in Q16 format (note the 16-bit shifts). The currently selected translation axis is shown in pink. It is important that the translation is done based on the screen's x,y,z axis, not the model's, as is done in the rotation tutorial. Table 3-2 provides the command keys for this tutorial.

Table 3-2 Command keys for the Translation tutorial

| Key | Action |
|------------|----------------------|
| Up arrow | Increase translation |
| Down arrow | Decrease translation |
| 1 | Select x axis |
| 2 | Select y axis |
| 3 | Select z axis |

At the top of Figure 3–2, you can see that vect (translation vector to set) is a variable. Users select which axis they want to translate along and increase/decrease the amount of translation along that axis.

The components of the translation vector are in Q16 format. This means the range for each component is (-65536 to 65536). The fact that you are shifting up 16 bits for each component means that you increase/decrease the translation by an integer amount every time.

Code example

The following is a code example for applying a fixed translation to a model and drawing it. Note that `I3D_StartFrame()` will need to be called to actually begin rendering the frame.

```
TranslateAndDraw( MyApp* pMe)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // pMe->pModel: Pointer to the model instance
    // pMe->m_p3DUtil: I3DUtil instance created on application init

    AEE3DTransformMatrix transform;
    AEE3DPoint vect = {0, -40<<16, 100 << 16};    // translate -40 in Y, and +100 in Z
    I3DUtil_SetIdentityMatrix(pMe->m_p3DUtil, &transform);
    I3DUtil_SetTranslationMatrix(pMe->m_p3DUtil, &vect, &transform);
    I3DModel_SetSegmentMVT(pMe->pModel, &transform, -1);
    I3DModel_Draw(pMe->pModel, pMe->m_p3D);
}
```

3.2 Projections

The projection tutorials in TutorI3D demonstrate the various I3D APIs that control how a scene is projected onto the display. There are four projection tutorials in TutorI3D:

- adjusting focal length
- adjusting the view depth
- adjusting the screen mapping
- setting a clipping rectangle
- The tutorials will affect one another, meaning that you can, for example, change the focal length first, then change the view depth and the screen mapping and view the overall result.

3.2.1 Focal length

The focal length tutorial allows you to increase or decrease the focal length of a scene. Increasing the focal length makes less of the scene apparent, and decreasing it makes more of a scene apparent. Objects that are placed at a depth equal to the focal length of the scene, are the ones that will appear to be most in focus, and will be more accurately rendered. The main API function utilized in this tutorial is `I3D_SetFocalLength()`. A screen shot of this tutorial is shown in Figure 3–3



Figure 3–3 Focal length tutorial screen

This tutorial provides a model in the center of the scene and allows you to adjust the focal length. The information screen provides the current value of the focal length. The two command keys for this tutorial are shown in Table 3-3.

Table 3-3 Command keys for the focal length tutorial

| Key | Action |
|------------|-----------------------|
| Up arrow | Increase focal length |
| Down arrow | Decrease focal length |

NOTE The range of the focal length is the same as the range for the z-buffer, that is 1 to 32767.

Code example

The following is example code for incrementing the current focal length by a specified amount. The new focal length will be applied to the next frame that is rendered.

```
IncrementFocalLength( MyApp* pMe,  int focal_increment)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // focal_increment: Amount to increment current focal length by

    uint32 focalLength;
    I3D_GetFocalLength(pMe->m_p3D, &focalLength)
    focalLength += focal_increment;
    Do range checking on focalLength (range is 1 to 32767)
    I3D_SetFocalLength(pMe->m_p3D, focalLength)
}
```

3.2.2 View depth

The view depth tutorial allows you to change the depth of the scene. The view depth is defined by two imaginary planes called the near clipping plane and the far clipping plane. Objects at a depth which is in between these two planes are visible and will be rendered, while objects closer to the viewer than the near plane or farther away than the far plane, will not be visible and will not be rendered. Setting an appropriate view depth for a scene is important because it determines the amount of depth detail that is present. Having a large view depth will allow you to render scenes with lots of depth and detail, at the cost of more computations to render the scene. The main API function utilized in this tutorial is `I3D_SetViewDepth()`. A screen shot of this tutorial is provided in Figure 3–4.

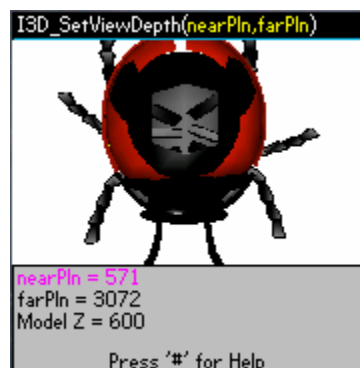


Figure 3–4 View depth tutorial screen shot

This tutorial allows you to select the near plane or the far plane and adjust the depth of that plane. In the information screen at the bottom, the current depth of the near and far plane is displayed, as well as the depth of the model in the scene. Increasing or decreasing the near or far plane through the model will cause only cross sections of the model to be displayed, as only the parts of the model within the view depth will be rendered. Notice that in Table 3-4, a portion of the model lies outside the view depth, and is therefore not rendered. Table 3-4 provides the command keys for this tutorial.

Table 3-4 Command keys for the view depth tutorial

| Key | Action |
|------------|-----------------------|
| Up arrow | Increase plane depth |
| Down arrow | Decrease plane depth |
| 1 | Select the near plane |
| 2 | Select the far plane |

NOTE The range of the view depth is 1 to 32767. This means the near and far plane can be anywhere in this range as long as the near plane is closer to the viewer than the far plane, closer a smaller number

Code example

The following is example code for incrementing the current value of the near plane and the far plane by a specified amount. The new focal length will be applied to the next frame that is rendered.

```
IncrementViewDepth( MyApp* pMe,  int nearPlaneIncrement, int
farPlaneIncrement)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // nearPlaneIncrement: Amount to increment the current near plane value by
    // farPlaneIncrement: Amount to increment the current far plane value by

    uint16 nearPlane, farPlane;
    I3D_GetViewDepth(pMe->m_p3D, &nearPlane, &farPlane);
    Do range checking on nearPlane + nearPlaneIncrement (range is 1 to 32767)
    Do range checking on farPlane + farPlaneIncrement (range is 1 to 32767)
    nearPlane += nearPlaneIncrement;
    farPlane += farPlaneIncrement;
    I3D_SetViewDepth(pMe->m_p3D, nearPlane, farPlane);
}
```


3.2.3 Screen mapping

The screen mapping tutorial allows you to change the scale and location of objects that are drawn into an I3D scene. Four parameters define the screen mapping in I3D. These are the x scale, the y scale, the x shift, and the y shift. The first two are the scaling factor for the scene. Whenever objects are drawn into the scene, the scaling factors are applied in the x direction and the y direction to increase or decrease the size of objects in the respective direction. In this manner, you can set the aspect ratio of the scene. A scaling factor of 1 in each direction means no scaling will be applied. The next two parameters, x shift and y shift, specify the amount of translation to be applied to every object drawn to the scene. Objects are usually centered at the point (0,0), meaning that when they are drawn, they will be placed at the point (0,0) in the I3D coordinate system. The point (0,0) in the I3D coordinate system is the top left of the display. Therefore, if x shift and y shift are 0, the model will be drawn at the top left corner of the screen. To specify the middle of the screen as the origin, you can make x shift and y shift the middle of the screen, then all models will be drawn in the center of the screen.

Important to note is that changing any of the screen mapping parameters changes it for the entire scene and not just for a single model.

The API function used to change the screen mapping is `I3D_SetScreenMapping()`. A screen shot of this tutorial is provided in Figure 3–5.

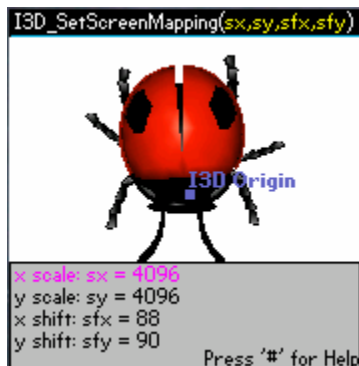


Figure 3–5 Screen mapping tutorial screen shot

The information screen at the bottom displays the current values of the four screen mapping parameters. Users are able to select and adjust any of the four parameters. The x scale and y scale parameters are in Q12 format, where 4096 means no scaling. The x shift and y shift parameters are specified in pixels. Table 3-5 provides the location of the new origin after applying the shift. The command keys for this tutorial are given in Table 3-5.

Table 3-5 Command keys for the screen mapping tutorial

| Key | Action |
|------------|-----------------------------------|
| Up arrow | Increase screen mapping parameter |
| Down arrow | Decrease screen mapping parameter |
| 1 | Select x scale |
| 2 | Select y scale |
| 3 | Select x shift |
| 4 | Select y shift |

Code example

The following is example code to set up the screen mapping for unit scaling and for setting the I3D origin to be the center of the screen.

```

SetupScreenMapping( MyApp* pMe)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // pMe->cxScreen: Width of the display screen
    // pMe->cyScreen: Height of the display screen.
    int xShift, yShift;

    //moves the origin to the center of the screen
    xShift = pMe->cxScreen/2;
    yShift = pMe->cyScreen/2;

    // 1 << 12 = 4096 (unit scaling)
    I3D_SetScreenMapping( pMe->m_p3D, 1<<12, 1<<12, xShift, yShift))
}

```

3.2.4 Clipping rectangle

The clipping rectangle is a 2D area in which the I3D scene is visible. All objects that fall within the bounds of the clipping rectangle will be rendered and visible. Everything outside of the clipping rectangle will not be rendered. Therefore, a scene with a large clipping rectangle will display more of the scene, but will take more time to render, and a scene with a small clipping rectangle will display less of the scene but will take less time. By default, the clipping rectangle is the entire screen. The API function used to change the current clipping rectangle is `I3D_SetClipRect()`. A screen shot of the clipping rectangle tutorial is given in Figure 3–6.



Figure 3–6 Clipping rectangle tutorial screen shot

In Table 3-6 we have the model in the middle of the scene, and a clipping rectangle which only allows a portion of the model to be visible. Note that normally the outline of the clipping rectangle will not be drawn. It's drawn here to facilitate visualization. The information screen shows the coordinate of the upper left corner of the clipping rectangle, as well as its width and height. The values are specified as pixels. Users can select and adjust any of the four rectangle parameters in this tutorial. The command keys for the tutorial are given in Table 3-6.

Table 3-6 Command keys for the clipping rectangle tutorial

| Key | Action |
|------------|---|
| Up arrow | Increase rectangle parameter |
| Down arrow | Decrease rectangle parameter |
| 1 | Select upper left x coordinate of rectangle |
| 2 | Select upper left y coordinate of rectangle |
| 3 | Select rectangle width |
| 4 | Select rectangle height |

NOTE The rectangle parameters are in pixels.

Code example

The following is example code to set the I3D clipping rectangle. The rectangle's upper left corner is 1/3 away from the left edge, and 1/3 down from the top. It's width and height are 1/4 of the width and height of the screen respectively.

```
SetupClippingRectangle ( MyApp* pMe)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // pMe->cxScreen: Width of the display screen
    // pMe->cyScreen: Height of the display screen.
    AERect clipRect;

    clipRect.x = pMe->cxScreen/3;
    clipRect.y = pMe->cyScreen/3;
    clipRect.width = pMe->cxScreen/4;
    clipRect.height = pMe->cyScreen/4;

    I3D_SetClipRect( pMe->m_p3D, &clipRect)
}
```

3.3 Lighting and materials

The lighting and materials tutorials in TutorI3D demonstrate the various I3D APIs that are used to set up lighting and material effects. The use of these APIs can make a big difference in adding realistic detail to a 3D scene. Two tutorials are used here to demonstrate these APIs. They are the lighting direction and color tutorial and the material tutorial.

3.3.1 Direction and color

The type of light used in I3D is directional light. This means the light source is at infinity, and heads along a specified direction vector. Whenever the light rays intersect an object, the rays are reflected in many different directions. The ones that reach the eye cause the object to light up. The color of the light determines how the object will light up. Light that hits a surface straight on will light a surface up more intensely than if it is at an angle, because much less scattering occurs on the reflected rays. Most of the reflected rays in this case will reach the eye. As the angle between the object surface and the light direction vector increase, the intensity of the light begins to diminish.

Different lighting models, based on the above principles, can be used to simulate very realistic effects. The two lighting models used in I3D are diffused light and specular light.

Diffused light is based on the principle that not all reflected light rays will reach the eye again. the parts of the object whos reflected rays do reach the eye should be lit up, and the intensity is based on the angle of the reflected rays. The parts of the object whos reflected rays do not reach the eye should not be lit up. Diffused light, falls off slowly across the object's surface, making a smooth transition between the parts that are lit and the parts that are not. As a result, diffused light will light up a fairly large portion of an object.

The specular light model is based on the second principle that light that hits a surface straight on, at no angle, will scatter much less. Therefore, most of the reflected rays will reach the eye, causing an intense bright spot on the object. This is known as a specular highlight. The intensity of this highlight falls off very rapidly across the surface of the object, so it only lights up a small region of the object.

Both diffused lighting and specular lighting can have their own direction and color. However, having both diffused and specular light, and each having their own color and direction, can be computationally intensive. For this reason, I3D has four different lighting modes as shown in Table 3-7.

Table 3-7 I3D lighting modes and properties

| Lighting mode | Description |
|-----------------------------------|---|
| Diffused | White diffused light; can only modify direction. |
| Color Diffused | Diffused light with color; can modify direction and color. |
| Diffused and Color Specular | White diffused light with color specular light. Can modify diffused direction and specular color. Specular direction is shared with diffused. |
| Color Diffused and Color Specular | Color diffused light with color specular light. Can modify diffused direction and color, and specular direction and color. |

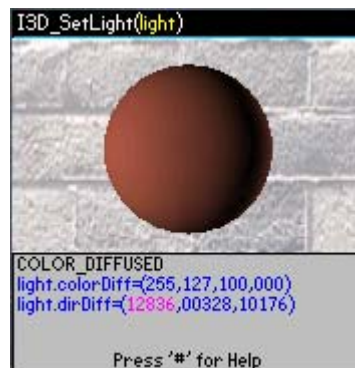
The lighting modes in Table 3-7 are in order from the least computationally intensive to the most intensive. The API function used to change lighting modes is `I3D_SetLightingMode()`, and the function for changing light settings is `I3D_SetLight()`. TutorI3D allows you to cycle between the different lighting modes described in Table 3-7, and change the direction and color of the diffused and specular lights and visualize the corresponding changes.

Figure 3–7(a-d) are screen shots of the tutorial showing the different lighting modes. Notice from Figure 3–7 how the different lighting modes affect rendering the model. In Figure 3–7a, we have only white diffused light. We can see that the left side of the ball is lit up while the right side is dark. Figure 3–7b is similar except now the diffused light has color. At the bottom of the information screen we can choose to change both the direction and color.

Figure 3–7c and d add specular lighting. We can see the specular highlight, and the detail it adds to the scene. Also, we can see how small the area of coverage for the specular light is. In the Diffused and Color Specular mode, seen in Figure 3–7c, the direction is shared between the diffused and specular light, so changing the diffused direction will also change the specular direction. In this mode the diffused light is white and cannot be changed, but the specular color can be changed. Figure 3–7d shows the Color Diffused and Color Specular mode, where each light has its own direction and color. The currently selected light type, diffused or specular, is highlighted in blue in the information screen. The currently selected parameter of the light type is highlighted in pink. For example, in Figure 3–7c, the blue color component of the specular light is selected. Users can increase/decrease the currently selected item. Table 3-8 below shows the command keys for this tutorial.



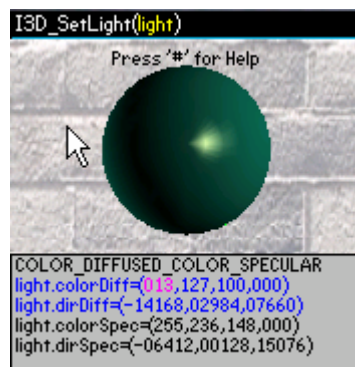
(a) Diffused



(b) Color Diffused



(c) Diffused and Color Specular



(d) Color Diffused and Color Specular

Figure 3–7 Lighting direction and color tutorial screen shots showing different lighting modes

Table 3-8 Command keys for the lighting direction and color tutorial

| Key(s) | Action |
|-----------|--|
| Up/down | Increase/decrease selected lighting parameter |
| (1,2,3) | (x,y,z) light direction component select |
| (4,5,6,7) | (red, green, blue, alpha) color component select |
| 8 | Toggle between diffused and specular light |
| 9 | Toggle the lighting mode |

The directional vectors for lighting are in Q14 format, meaning the range is from (-16384 to 16384) for each component. Also, the vector must be a unit vector, so increasing the x direction vector may also reduce the value of the y and z components to make the overall vector a unit vector (a unit vector is a vector where the magnitude is equal to 1). The color components are in the range (0 to 255).

Code example

The following is an I3D code example for setting the lighting mode to color diffused and color specular, as well as setting up individual parameters for diffused and specular lighting.

```

SetupLighting ( MyApp* pMe)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // pMe->m_ p3DUtil: I3DUtil instance created on application init

    AEE3DPoint direction;
    AEE3DLight light_value;
    AEE3DColor colorDiffused = {255,127,100,0};    // diffused color
    AEE3DColor colorSpecular = {220,220,220,0};    // specular color
    AEE3DPoint srcDirection = {0,0,16384};        // point into the center of
the scene

    I3D_SetLightingMode(pMe->m_p3D,
AEE3D_LIGHT_MODE_COLOR_DIFFUSED_COLOR_SPECULAR);

    // Make the direction vector a unit vector.
    //GetUnitVector returns a Q12 unit vector. Shift up 2 bits
    //to get a Q14 unit vector as SetLight expects
    I3DUtil_GetUnitVector(pMe->m_p3DUtil, &srcDirection, &direction);
    direction.x <=> 2;
    direction.y <=> 2;
    direction.z <=> 2;

    // set the diffused light for the scene.
    light_value.color = colorDiffused;
    light_value.direction = direction;

```

```
1      light_value.type = AEE3D_LIGHT_DIFFUSED;
2      I3D_SetLight(pMe->m_p3D, &light_value);
3
4      // set the specular light for the scene (use same direction as specular)
5      light_value.color = colorSpecular;
6      light_value.type = AEE3D_LIGHT_SPECULAR;
7      I3D_SetLight(pMe->m_p3D, &light_value);
8  }
9
```

3.3.2 Material

The material of an object determines the various visual aspects of that object. For example, the color of the object's surface, how much light the object reflects, the amount of light emitted by the object, and so on. I3D has three material properties that can be modified. They are: color, shininess, and emissive.

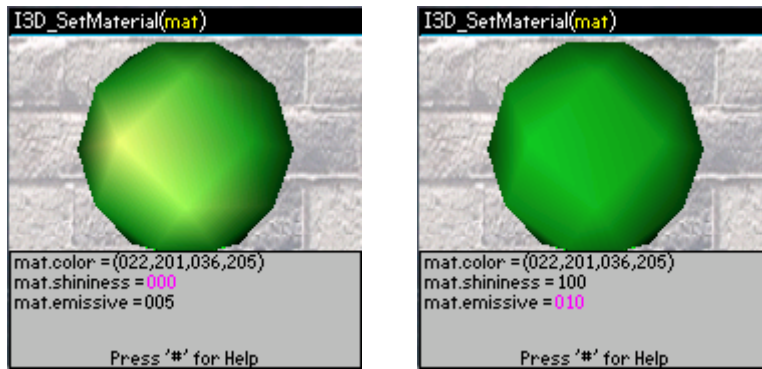
The color of an object's material determines the surface color of the object. Also, it determines the color of light that will light up the surface. For example, if we apply a blue material to an object, the object will only light up when a blue light is applied to it. A red light will not light up the object.

Shininess determines the amount of light that is reflected by the object as opposed to being absorbed. It can give an object a metallic look or a more dull look.

The emissive of an object determines the amount of light the object naturally emits. This is independent of any light sources in the scene. This can give an object a glowing look.

The API function used in I3D to set material properties is `I3D_SetMaterial()`. Screen shots of the TutorI3D material tutorial shown in Figure 3–8a provides a green material with high shininess and emissive. Note that a low value for shininess means high shininess. Figure 3–8b provides the same color material as Figure 3–8a, but, is more dull looking because of the shininess value.

Important to note here, is that the material color set by calling `I3D_SetMaterial()` only affects subsequent calls to `I3D_RenderTriangles`, `I3D_RenderTriangleFan()`, and `I3D_RenderTriangleStrip()`. `I3DModel_Draw()` is unaffected by this color because Q3D models have their own material colors. The emissive and shininess set by `I3D_SetMaterial()`, affects calls to `I3DModel_Draw(,)` as well as the other render functions.



(a)

(b)

Figure 3–8 Material tutorial screen shots. (a) shiny green material (b) dull green material

TutorI3D allows users to select any of the material parameters and increase/decrease them. The information screen provides the current values of the material color, shininess and emissive. The currently selected material parameter is highlighted in pink. The command keys for this tutorial are given in Table 3-9.

NOTE The range for all the material parameters is (0 to 255).

Table 3-9 Command keys for the material tutorial

| Key(s) | Action |
|-----------|--|
| Up/Down | Increase/decrease selected material parameter |
| (1,2,3,4) | (red, green, blue, alpha) color component select |
| 5 | Shininess select |
| 6 | Emissive select |

Code example

The following is an I3D code example for setting the material properties. Subsequent calls to `I3D_RenderTriangles`, `I3D_RenderTriangleFan()`, and `I3D_RenderTriangleStrip()` will be affected by all these material properties. Subsequent calls to `I3DModel_Draw()` will be affected by the emissive and shininess values set here.

```

SetupMaterial(MyApp* pMe)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init

    AEE3DMaterial material;

    material.color.r = 22;
    material.color.g = 200;
    material.color.b = 36;
    material.shininess = 5;
    material.emissive = 10;
    I3D_SetMaterial(pMe->m_p3D, &material)
}

```

3.4 Textures and blending

The textures and blending tutorials in TutorI3D demonstrate the I3D APIs that are used to apply textures to objects, including the perspective correction capability, and also demonstrates the alpha blending capability. Three tutorials are included in this section: texture rendering, alpha blending, and perspective correction.

3.4.1 Texture rendering

A texture is simply an image that can be applied to the surface of 3D objects. Different textures can be applied to the same objects in order to give them a specific look. For example, a wood texture can be applied to a cube to make it look like a wooden crate, or a marble texture can be applied to the same cube to make it look like a marble crate. A popular method of putting water into a 3D scene is to apply a water texture to a wide and thin surface.

If the surface of the object is not directly facing the viewer, or the object is placed far into the scene for example, then the texture must be interpolated to the proper dimensions and orientation before being applied. I3D uses the “nearest” method for interpolation. In this method, the pixel in the source texture nearest to the pixel in the destination texture (by location) is copied to the pixel in the destination texture. Another, more computationally intensive method of interpolation, would be to blend a number of source pixels to get a single destination pixel.

What if the size of an object’s surface is larger than the size of the texture? In this case, the texture needs to be wrapped before being applied to the surface. One example of texture wrapping is to repeat the texture and apply this new texture to the object’s surface. I3D provides a number of wrap methods, as shown in Table 3-10.

Table 3-10 I3D texture wrap modes

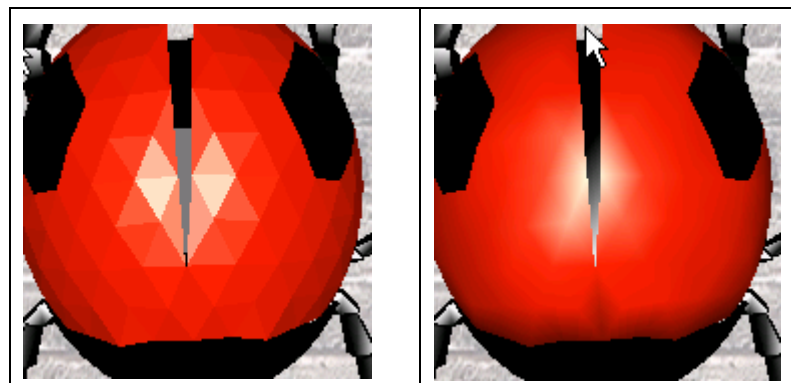
| Wrap mode | Description |
|-----------|--|
| Repeat | The original texture is repeated and applied to the object |
| Mirror | The original texture is mirrored and applied to the object |
| Clamp | Pixels outside the texture range use the border pixel of the original texture. |
| Border | Pixels outside the texture range use a predefined color |

Another important question to answer, is how are the vertex colors taken into account when textures are applied to an object's surface? There are numerous schemes to combine the object's surface colors with the texture colors. In I3D, you specify how they are combined using the render modes, which are described in Table 3-11. The render mode determines how to color the object using the vertex colors, and also how to combine these with the texture colors.

Table 3-11 I3D render modes

| Render mode | Description |
|-----------------------------|---|
| Flat Shading | Each triangle is filled with the same color as the color of the first vertex. The texture colors are not used |
| Flat Texture Fast Shading | Flat shading for surface colors, then average the surface colors with the texture colors |
| Flat Texture Shading | Flat shading for the surface colors, then blend the surface colors with texture colors |
| Smooth Shading | Each triangle is filled with the colors interpolated across all three vertices |
| Smooth Texture Fast Shading | Smooth shading for the surface colors, then average the surface colors with the texture colors |
| Smooth Texture Shading | Smooth shading for the surface colors, then blend the surface colors with the texture colors. |
| Texture Replace | Only texture colors used. Surface colors are not used |

The render mode plays a significant role in determining the speed of rendering. Flat shading is very fast, but low quality, while smooth shading is slower, because of the interpolations, but higher quality. Figure 3-9 provides the same 3D object rendered with both flat shading and smooth shading. In Figure 3-9a, each triangle has only one color, which allows us to see the rendered triangles. In Figure 3-9b however, the color within each triangle is interpolated across the 3 vertices, giving a much smoother surface and yielding a higher quality render.



(a) Flat Shading

(b) Smooth Shading

Figure 3-9 3D object rendered with (a) flat shading and (b) smooth shading

The I3D API function for setting the render mode is `I3D_SetRenderMode()`, and the API function for setting a texture is `I3D_SetTexture()`. These API functions will specify the render mode and texture for subsequent calls to `I3D_RenderTriangles()`, `I3D_RenderTriangleStrip()`, and `I3D_RenderTriangleFan()`. Objects drawn using `I3DModel_Draw()` have shading and textures specified as part of the model data, so these objects are not affected by `I3D_SetRenderMode()`, or `I3D_SetTexture()`.

The texture rendering tutorial of TutorI3D draws a textured 3D model, and allows user to change the render mode, the wrap mode, texture image itself, and to view the results. Table 3-10 provides a screen shot of this tutorial using a wood texture applied to a cube.



Figure 3–10 Texture rendering tutorial screen shot

I3D limits the width and height of the texture image to be a power of 2 no greater than 256. The bit depth of texture images is limited to 8 bits per pixel.

The command keys for this tutorial are shown in Table 3-12.

Table 3-12 Command keys for the texture rendering tutorial

| Key(s) | Action |
|--------|---|
| 1 | Change the Wrap_s mode. This is the wrapping mode in the horizontal direction |
| 2 | Change the Wrap_t mode. This is the wrapping mode in the vertical direction |
| 3 | Change the texture image |
| 4 | Change the render mode |
| 5 | Change the texture image |

Code example

The following is an I3D code example for setting the render mode to “texture replace” and for loading a texture from a resource file and setting it as the current texture for rendering. Subsequent calls to `I3D_RenderTriangles`, `I3D_RenderTriangleFan()`, and `I3D_RenderTriangleStrip()` will be affected by render mode and the new texture. Subsequent calls to `I3DModel_Draw()` will not be affected, as these models have their own shading properties and textures.

```
SetupTexture(MyApp* pMe)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init

    AEE3DTexture texture;

    // set the render mode to texture replace. Vertex colors will not be
    used.
    I3D_SetRenderMode(pMe->m_p3D, AEE3D_RENDER_TEXTURE_REPLACE);

    //setup the texture properties.
    texture.type = AEE3D_TEXTURE_DIFFUSED;
    texture.SamplingMode = AEE3D_TEXTURE_SAMPLING_NEAREST;
    texture.Wrap_s = AEE3D_TEXTURE_WRAP_REPEAT;
    texture.Wrap_t = AEE3D_TEXTURE_WRAP_REPEAT;
    texture.BorderColorIndex = 255;
    texture.pImage = ISHELL_LoadResBitmap(pMe->a.m_pIShell, RES_FILE_ID,
    MY_TEX);

    if(texture.pImage)
    {
        // set the texture and release since I3D_SetTexture() will increase
        the ref. count
        I3D_SetTexture(pMe->m_p3D, &texture);
        IBITMAP_Release(texture.pImage);
    }
}
```

3.4.2 Alpha blending

Alpha blending can give the appearance that 3D objects are transparent/translucent. This is accomplished by performing a weighted average of background pixels and the object's pixels to get the final pixel value. The alpha value of the object determines the weight of the components. Changing the weights allows the background or the object to be more dominant in the final pixel value. In this manner, objects can have different degrees of translucency, from completely transparent to fully opaque. In I3D, alpha blending is a capability that must be enabled using `I3D_Enable(AEE3D_CAPABILITY_BLENDING)`. This is because rendering with alpha blending requires more computations.

The alpha blending tutorial in TutorI3D allows users to change the alpha value of a model and see how this affects its appearance when drawn in front of a background. Figure 3-11 provides screen shots of this tutorial with various levels of alpha for the object, and Table 3-13 provides the command keys.

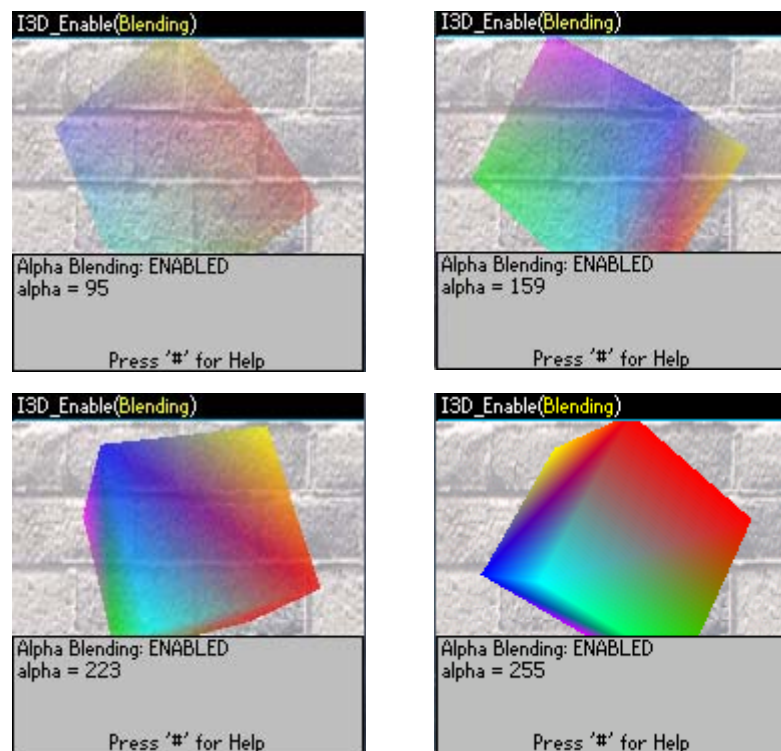


Figure 3–11 Screenshots of the alpha blending tutorial showing various levels of alpha

Table 3-13 Command keys for the alpha blending tutorial

| Key(s) | Action |
|---------|-------------------------------|
| Up/down | Increase/decrease alpha value |
| 1 | Enable/disable Alpha blending |

NOTE The range of alpha values is (0 to 255), however, I3D only supports five different levels of alpha, meaning the range is broken up into levels. The levels are shown below in Table 3 14.

Table 3-14 Alpha value levels, ranges, and number of values in range

| Level | Range of values | Number of values in range |
|------------------------|-----------------|---------------------------|
| Level 0: (transparent) | 0 to 31 | 32 |
| Level 1: | 32 to 95 | 64 |
| Level 2: | 96 to 159 | 64 |
| Level 3: | 160 to 223 | 64 |
| Level 4: (opaque) | 224 to 255 | 32 |

Code example

The following is an I3D code example for enabling alpha blending, and setting the alpha value for a model. Notice that the code below is modifying the model data. Because of this, it's very important that the code below be executed either before or after a frame render, and not during a frame render. Read the section about I3D events in the I3D API guide to learn about when it's appropriate to modify model data.

```

SetAlpha(MyApp* pMe)
{
    // pMe: pointer to application instance structure
    // pMe->m_p3D: I3D instance created on application init
    // pMe->model: model data

    int i;
    int alphaValue = 128;
    I3D_Enable(AEE3D_CAPABILITY_BLENDING);

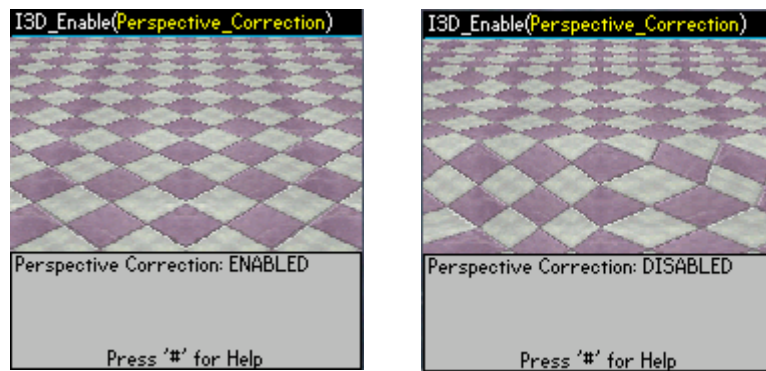
    for(i=0; i < pMe->model->NumVertices; i++)
    {
        // change alpha for all vertices in the model
        pMe->model->Vertex[i].alpha = alphaValue;
    }
}

```


3.4.3 Perspective correction

Perspective correction is a method that takes the depth (Z coordinate) of an object in 3D space into account when applying a texture to it. Without perspective correction, textured objects will appear to shift and tear in an unrealistic way. In I3D, perspective correction is a capability that is enabled by calling `I3D_Enable(AEE3D_CAPABILITY_PERSPECTIVE_CORRECTION)`.

The perspective correction tutorial in TutorI3D, provides a textured object drawn with perspective correction enabled, and allows the user to disable perspective correction in order to see how it affects rendering the textured object. Figure 3–12 provides screen shots of this tutorial with perspective correction enabled and disabled. Notice how the texture “tears” when perspective correction is disabled.



(a) Enabled

(b) Disabled

Figure 3–12 Screen shots of the perspective correction tutorial showing perspective correction (a) enabled and (b) disabled

The only command key for this tutorial is the 1 key, which will enable/disable perspective correction.

Code example

To enable perspective correction simply call:

```
I3D_Enable(AEE3D_CAPABILITY_PERSPECTIVE_CORRECTION)
```

All subsequent texture mapping will take perspective into account.

3 TutorI3D Tutorials