Computer Graphics: An Interactive Introduction with OpenGL

Dalton R. Hunkins
Computer Science Department
St. Bonaventure University
St. Bonaventure, NY 14778
hunkins@sbu.edu

Copyright © 2009 Dalton R. Hunkins
Published by Computer Graphics Materials Source (CGEMS)

Terms of Use

This suite of tutorials and examples is free software; you can redistribute them and/or modify them under the terms of the GNU General Public License, version 2 as published by the Free Software Foundation (see either of the files GNU GPL v2.docx or GNU GPL v2.pdf). There are many more files included in this distribution however some are being re-distributed here for your convenience (see the Third-Party Files section).

This suite of tutorials and examples are distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License, version 2 for more details.

Third-Party Files

The two files cygwin1.dll and glut32.dll are included with the installation of Cygwin and have been downloaded from http://www.cygwin.com/. They are being re-distributed here under the terms of the Cygwin license and, in particular, the GNU General Public License, version 2. Under those terms, the cygwin source code is available at http://cgems.inesc.pt/resources/Cygwin_Source.zip.

The three Wavefront files f-16.obj, porsche.obj and rose+vase.obj and their associated materials files (.mtl files) that reside in the resources directory ComputerGraphicsTutorials/Resources/Wavefront can be found with Nate Robbins's tutorials and downloaded from http://www.xmission.com/~nate/tutors.html. There are no stated terms of use and no license agreement given at this site.

The six clip art files target0.bmp, target1.bmp, target2.bmp, target3.bmp, target4.bmp and weapon.bmp that reside in the directory ComputerGraphicsTutorials/Resources/ImageMaps/ShootingGarme were formed from png files that are in the Public Domain and downloaded from http://www.pdclipart.org/.

Abstract: *Computer Graphics: An Interactive Introduction with OpenGL is a suite of examples and interactive tutorials. The software is created for the person who wants an introduction to computer graphics using the OpenGL API. It is assumed that the reader is familiar with programming in the C++ language; knowledge of OpenGL is not a pre-requisite. Also, the tutorials and examples may be used in conjunction with many of the mainstream books on computer graphics.*

Keywords: Computer Graphics, OpenGL, Interactive Tutorials

Table of Contents

Introduction

Computer Graphics: An Interactive Introduction with OpenGL is a suite of examples and interactive tutorials written in C++ that uses OpenGL. The software has involved over a period of time. Some of the tutorials were first written using the Ada binding to OpenGL. This subset was introduced in a paper presented at SIGGRAPH 2001 [Hunkins and Levine, 2001]. After some modifications including the addition of several more tutorials, the revised Ada version was presented through another paper published in *Computer and Graphics* [Hunkins and Levine, 2002]. Part of the evolution included the tutorials being written with JOGL, the Java binding to OpenGL; this work has not been published. The current suite has been used several semesters by computer science majors in a computer graphics course taught by the author. Although inspired by Nate Robbins' tutorials (http://www.xmission.com/~nate/tutors.html), the software has been independently developed. You will find, however, three Wavefront models (obj files) from his downloaded files are used in the tutorials. The three models are included in the download for convenience but are not part of the author's copyright.

In addition to the tutorials, the current suite now includes many examples with the source code displayed at runtime with online annotations. The original goal of providing an introduction to many of the fundamental concepts in computer graphics has been preserved with the current version. The concepts include basic geometry and transformations, two-dimensional orthographic projections, color models, viewing through the virtual camera, lighting and illumination models, and texture mappings. The current version also contains a set of examples that introduce game programming.

The software suite is not only built on top of OpenGL but it also uses the GLUT and GLUI libraries. With the work being an introduction to computer graphics, the OpenGL features covered are those of version 1.2 (and, of course, available in version 2.1). But with the publication of the standard for OpenGL, version 3.0, one may ask why educational software built on earlier versions is still viable. One of the major differences between the versions is the "Depreciation Model" presented in version 3.0. In particular, many features (functions) that have been used in an introductory course such as the construction of geometries using the glBegin-glEnd pair are marked as deprecated in 3.0. Although they have not been removed, it is stated that deprecated items will no longer exist in future versions of OpenGL. Until then and until computer graphics educators and authors of graphics textbooks adapt to the change, earlier versions of OpenGL are still useful. Further, Nvidia states on its website (http://developer.nvidia.com/object/opengl_3_driver.html) that "Nvidia has *no* plans for dropping support for OpenGL 2.1, and earlier versions, on our existing *and* future shipping hardware."

Installing the Software

The tutorials and examples were developed on a Windows XP platform and ported to Windows Vista; for now, Windows is the only platform on which they are available. Installation of the software simply requires unzipping the provided file. After which, you must copy the three dll files utilsWin32.dll, glut32.dll and cygwin1.dll that are in the extracted folder Libraries\DLLs to a folder that is in the search path for Windows. For example, you may copy the files to the Windows system directory c:\Windows\system32. Of course, you can instead set the Path variable to include the above DLL folder.

The library utilsWin32.dll (referred to as Utils) was created by the author and provides resources needed by the tutorials. The IDE used to create the Utils library, the tutorials and the examples is Bloodshed's Dev-C++ (http://www.bloodshed.net/devcpp.html) with the C++ compiler provided by Cygwin. Cygwin (http://www.cygwin.com/) is a Linux like environment for Windows. The Cygwin software is free and as noted above, the two dll files cygwin1.dll and glut32.dll are being redistributed here under the terms of the GNU General Public License. The version of GLUT (glut32.dll as provided here) came as part of the Cygwin download. Although other versions are available (see for example, http://www.opengl.org/resources/libraries/glut/), the glut file provided here must be used.

You must also have OpenGL installed on your machine. These are provided through two dlls, opengl32.dll and glu32.dll, which are normally installed with the Windows operating system. (Look for them in the folder c:\Windows\system32).

The tutorials and examples use Internet Explorer for viewing the ancillary materials. It is assumed that IE is installed in the normal Windows location, namely, c:\Program Files\Internet Explorer\iexplore.exe. Also the vbs script interpreter cscript.exe is used in closing IE through a menu choice; it is assumed that it resides in c:\Windows\system32.

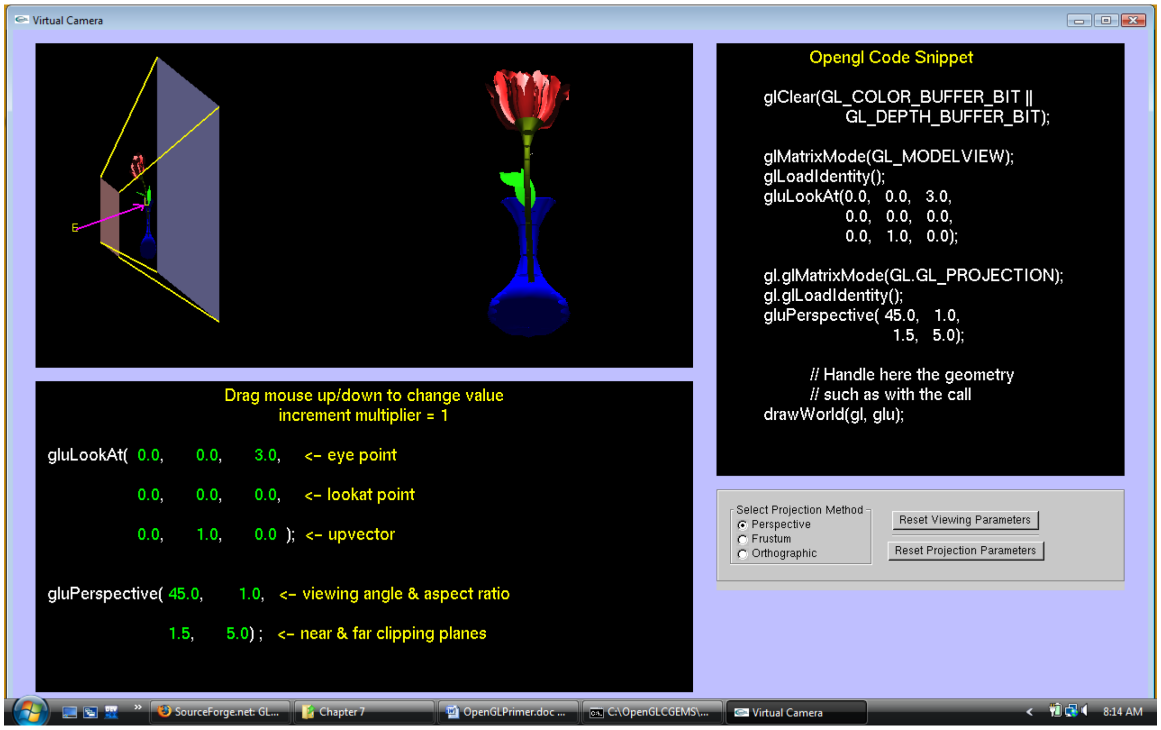
Last, if you choose to copy the software to another drive or location, you must preserve the contents of the ComputerGraphicsTutorials folder – that is, do not change any of its file structure and names.

Using the Launch Program

The program Launch.exe, which resides in the ComputerGraphicsTutorials folder at its root, is provided for convenience. You can open (run) any of the tutorials and examples by running this program. . When it executes, you will see a page giving brief descriptions of the tutorials and examples. The page may be scrolled by dragging the mouse up or down with the left button down or by using the up and down arrow keys. The program also has a popup menu that is triggered with a right mouse click. The submenus, which are part of the popup menu, provide choices of tutorials and examples you can execute. Simply click on any of the selections. Obviously, you can browse to a folder containing an executable and run it directly.

Running A Tutorial

The tutorials are constructed with consistency in mind. Each tutorial is an executable (.exe) file. When executed, you will be viewing a window consisting of several sub windows. Each tutorial has at least one sub window containing a graphics image constructed through the use of OpenGL. Other sub windows provide interactive controls and information pertaining to the tutorial. A screen shot of one such tutorial is shown in figure 1.



Opengl Code Snippet

```
glClear(GL_COLOR_BUFFER_BIT ||
        GL_DEPTH_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0,   0.0,   3.0,
          0.0,   0.0,   0.0,
          0.0,   1.0,   0.0);

gl.glMatrixMode(GL.GL_PROJECTION);
gl.glLoadIdentity();
gluPerspective( 45.0,   1.0,
                1.5,    5.0);

    // Handle here the geometry
    // such as with the call
drawWorld(gl, glu);
```

Drag mouse up/down to change value
increment multiplier = 1

```
gluLookAt( 0.0,   0.0,   3.0,    <-- eye point

           0.0,   0.0,   0.0,    <-- lookat point

           0.0,   1.0,   0.0 );  <-- upvector


gluPerspective( 45.0,   1.0,   <-- viewing angle & aspect ratio

                1.5,    5.0) ;  <-- near & far clipping planes
```

Select Projection Method: [✓ Perspective] Frustum Orthographic

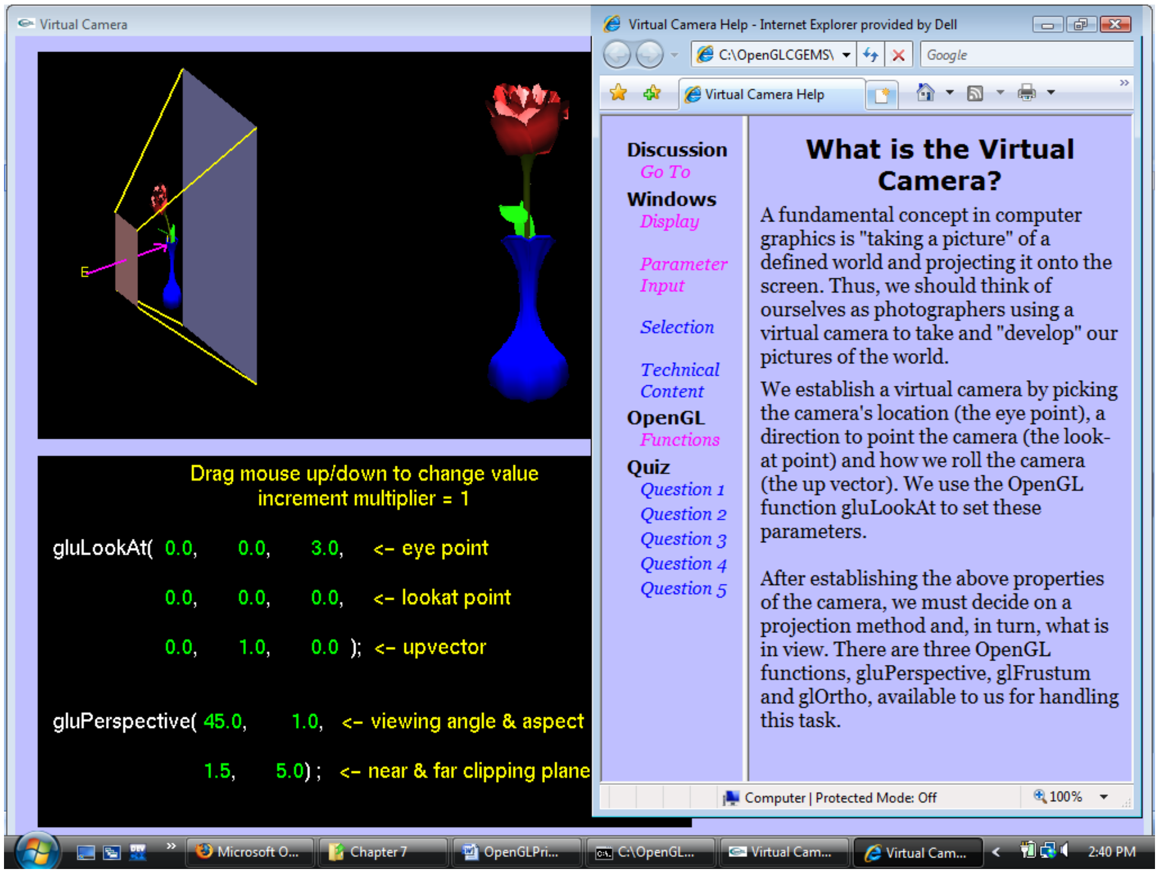
Figure 1 Virtual Camera Tutorial

A popup menu with three choices, “Show Help”, “Close Help” and “Close Tutorial”, is also part of each tutorial. The menu is activated by right clicking the mouse in a portion of the main window that is not part of any of the sub windows.

Selecting “Show Help” brings up Internet Explorer, which in turn allows you to view several html pages. Figure 2 is a screen shot of the tutorial along with the Internet Explorer window that opens with the “Show Help” menu choice.

The choice “Close Help” simply closes Internet Explorer and the display of the help pages. Of course, you can re-open the help pages anytime with the “Show Help” menu choice.

The “Close Tutorial” option is for closing the program versus using the standard method for closing a window. Closing the tutorial through this menu choice facilitates any memory management built into the tutorial. Further, this choice also closes Internet Explorer and the help pages if it is open.



```
        Drag mouse up/down to change value
              increment multiplier = 1

gluLookAt( 0.0,    0.0,    3.0,   <- eye point

           0.0,    0.0,    0.0,   <- lookat point

           0.0,    1.0,    0.0 ); <- upvector


gluPerspective( 45.0,    1.0,   <- viewing angle & aspect

                1.5,     5.0) ;  <- near & far clipping plane
```

What is the Virtual Camera?

A fundamental concept in computer graphics is "taking a picture" of a defined world and projecting it onto the screen. Thus, we should think of ourselves as photographers using a virtual camera to take and "develop" our pictures of the world.

We establish a virtual camera by picking the camera's location (the eye point), a direction to point the camera (the look-at point) and how we roll the camera (the up vector). We use the OpenGL function gluLookAt to set these parameters.

After establishing the above properties of the camera, we must decide on a projection method and, in turn, what is in view. There are three OpenGL functions, gluPerspective, glFrustum and glOrtho, available to us for handling this task.

Figure 2 Virtual Camera with Show Help

As seen in figure 2, a list of links appears on the left of the IE window. The *Discussion* link returns you to the page that is initially opened and gives you an introduction to the concept being addressed by the tutorial. There are other links under the title *Windows*. Each of these links brings up a page that describes the content of the sub window as it pertains to the tutorial along with any interactive controls associated with the sub window. The link *OpenGL Functions* opens a page showing the functions that are presented through the tutorial. Most tutorials even include a topic *Quiz* with links to questions that students may use as a self-check of their understanding of the material.

Any of the ancillary materials can be easily modified locally since they are presented in html format. For example, suppose you want to change or add to the quiz questions. One needs only open Links.html in the appropriate “HelpFiles” folder (e.g. ComputerGraphicsTutorials/Resources/HelpFiles/VirtualCamera) and change the links to questions. Also, since the questions are also in html format, new ones can be added with any of the existing ones serving as a template.

The tutorials reside in Tutorials, which is a subfolder of ComputerGraphicsTutorials. Further, the executable files are organized into chapter subfolders. For example, the executable VirtualCamera.exe is in the folder ComputerGraphicsTutorials\Tutorials\Chapter 7.

You should run the Virtual Camera tutorial so as to become familiar with the general layout used in any of the tutorials. What you see on the screen when running the tutorial should agree with the screen shot shown in figure 1. It should be noted that sub windows that contain textual material are constructed through the use of GLUT’s fixed font text and in turn are not scalable. Thus, if a portion of the text does not appear within the sub window, then, if possible, increase your screen resolution. (The tutorials and examples were developed on a laptop with screen resolution 1280 by 800; the resolution of 1024 by 768 is also reasonable.)

Every tutorial produces at least one rendered image that is displayed in a sub window. We call this type of sub window a ***display window*** (the content of these windows do scale***)***. For the Virtual Camera tutorial, the display window in the upper-left corner has two parts (viewports); on the right is a rose in a vase and to its left is the same object but shown within what is called the view volume of the virtual camera.

The tutorials will have one or more other sub windows that provide interactive controls. For the Virtual Camera tutorial, the sub window in the bottom-right corner contains radio buttons and push buttons. We call this type of sub window a ***selection panel***. The controls in a tutorial's selection panel were constructed from the GLUI User Interface Library that was built by Paul Rademacher and is currently maintained by Nigel Stewart and Bill Baxter (http://sourceforge.net/projects/glui).

The sub window in the bottom-left corner displays another means of interaction. The sub window, which we call a ***parameter input window***, shows two OpenGL functions that are particular to the tutorial. In any parameter input window, you can change the values that are drawn in green. To change a parameter, move the mouse over the parameter; press and hold down the left mouse button. The color of the selected parameter changes to yellow, indicating the value is being changed. The value changes while dragging the mouse up and down. You may also change a selected parameter through use of the up and down arrow keys. A factor is associated with this method. The factor multiples the increment/decrement used with each key stroke. The factor in use is shown at the top of the parameter input window. The factor may also be increased or decreased by pressing the Page-Up or Page-Down keys. Last, a parameter input window may be scrolled by dragging the mouse with its right button pressed.

Depending on the tutorial you are running, there may be other sub windows within the main tutorial window. For the Virtual Camera tutorial, a C++ code snippet is shown in the upper-right sub window. The code snippet is updated as you change values in the parameter input window. Other tutorials may have a sub window showing mathematical computations used in the tutorial. A sub window that displays a code snippet or mathematical computations or other types of information is called a ***technical content window***. Many of these information windows are scrollable. The method for scrolling is the same as for a parameter input window.

The general structure and layout exhibited with the Virtual Camera tutorial is consistently used throughout the tutorials.

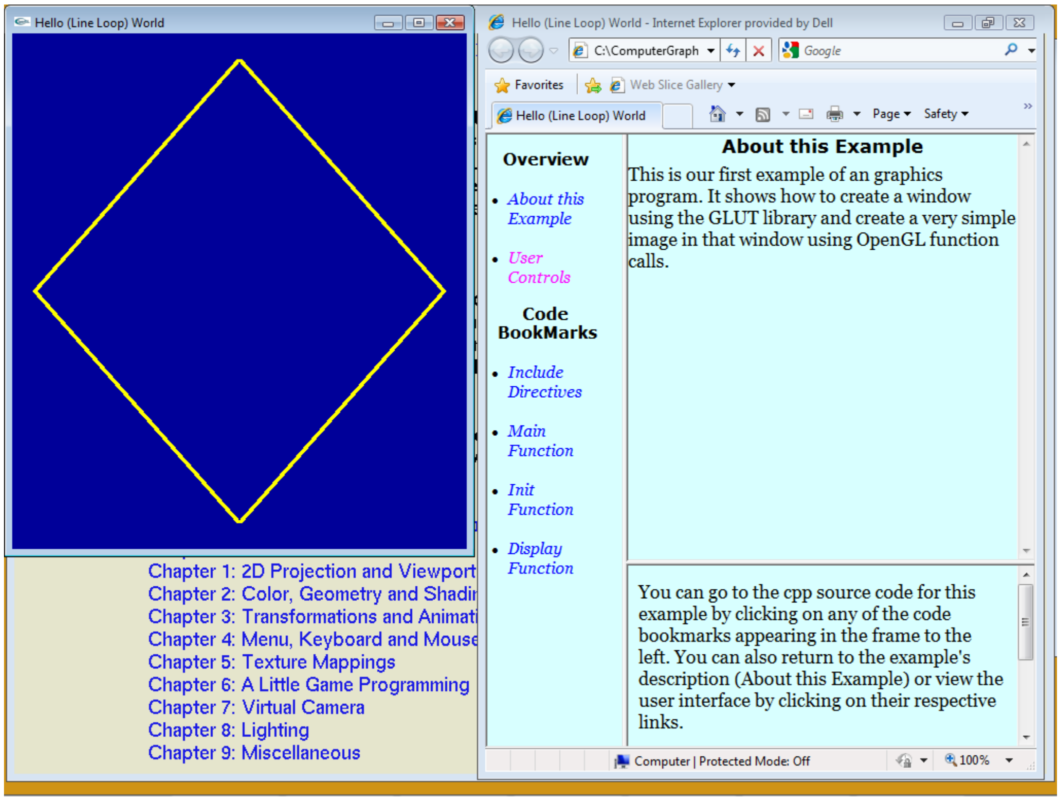
Running an Example Program

Each chapter has at least one section containing a complete C++ program. The program builds on the previous material and uses OpenGL, GLUT and possibly GLUI functions. The examples are contained in the sub folder Examples under the main folder ComputerGraphicsTutorials. The executables for the examples are also organized by chapters. Similar to running the tutorials, you can run the examples either directly or through the launch program.

A screen shot of the Hello World example is shown in figure 3. Two windows appear on the screen. The window on the left is the GLUT window created by the program while the window on the right is an instance of Internet Explorer. Each of our example programs will automatically invoke IE to display html documents. The documents include a list of bookmarks and/or links on the left. The content of the frame on the right is either a brief description of the example, a description of the user controls for the example or a code listing of the program (or a portion thereof) that created the window and its OpenGL content. Of course, the particular display in the right frame is dependent on the selected link or bookmark.

Also, selected lines of code in the listing are linked to other html documents (code displayed in light blue text indicates a link). A discussion of the code's purpose will appear in the frame below the code listing when you click on the link.

A popup menu that is right mouse triggered is also part of the program example. The minimum choices are "Show Source Code", "Close Source Code" and "Close Example." The first choice is provided so that the code listing and discussion may be re-opened in case you close the one that automatically opened. The third menu choice not only closes the program but it also closes Internet Explorer and its display of the code listing and ancillary materials.



Chapter 1: 2D Projection and Viewport
Chapter 2: Color, Geometry and Shadi
Chapter 3: Transformations and Animati
Chapter 4: Menu, Keyboard and Mouse
Chapter 5: Texture Mappings
Chapter 6: A Little Game Programming
Chapter 7: Virtual Camera
Chapter 8: Lighting
Chapter 9: Miscellaneous

Overview

- *About this Example*
- *User Controls*

Code BookMarks

- *Include Directives*
- *Main Function*
- *Init Function*
- *Display Function*

About this Example

This is our first example of an graphics program. It shows how to create a window using the GLUT library and create a very simple image in that window using OpenGL function calls.

You can go to the cpp source code for this example by clicking on any of the code bookmarks appearing in the frame to the left. You can also return to the example's description (About this Example) or view the user interface by clicking on their respective links.

Figure 3 Hello World Example with Annotated Source Code

Comments on Use of Software

As noted in the introduction, the suite of tutorials and examples has evolved over the past 10 years. Also, the current version has been used for several semesters in an introductory computer graphics course taught by me. I am sharing herein comments on how it was used in that course.

The course, CS 256 taught at St. Bonaventure University, consists of a “lecture” portion, meeting 2.5 hours per week, and a closed lab portion, meeting 2 hours per week. The suite is used in both components in various ways.

It has been said that a picture is worth a thousand words but I add that it is worth even more when it is interactive. Further, it is understood that students learn best when they are active participants in the process. Thus, when I used the software in a “lecture”, I asked a student to be the user and “operate” the tutorial and/or example. But, of course, the student is guided in her exploration with a script that I provide either verbally or in written form. Appendix B contains samples of three (abbreviated) scripts. The scenarios, which I call Discovery Activities for lack of a better term, are designed for the students, including the designated user, to understand through exploration a specific concept and/or specific API function. For example, activities in the Color Models script are written to discover (understand) saturation in the HSV model and the effect occurring in the RGB model when changes are made to the color’s saturation. In turn, it is hoped that the student develops an understanding of the relationship between the two models. The scenarios easily extend during a “lecture” beyond the ones I initially provide since all students are encouraged to suggest other activities.

Assignments in the closed labs may take one of two forms. One form is a script of Discovery Activities and the student is required to submit her recorded observations. Another form is the somewhat traditional "write a program that …." Lab assignments may also be a combination of the two. For example, the first closed lab in CS 256 pertains to the Hello World program. The first sample script in Appendix B suggests part of the assignment; the other part requires the student to submit a modified Hello World program. Obviously, the first lab serves as an "ice breaker" and, in turn, is not overly demanding. With respect to labs that strictly require programming, the assignments are either extensions of the examples or are programs requiring development "from the get-go" and are based on concepts presented in the tutorials. The latter type of lab assignment indicates why I have chosen to have example software and tutorials in the suite. The program examples are just that with the entire source code visible to the student. On the other hand, a tutorial focuses on a concept with its source code not being pertinent to the learning objective. I may give, for example, a lab assignment that requires the student to incorporate the viewing transformation within her program. Or I may ask that a program be written that produces a shadow on a plane that is not one of the coordinate planes. For this latter example, the Projection Shadow tutorial gives the student the necessary information on computing the projection transformation.

I hope that you find the suite of tutorials and examples useful in these and other ways when teaching an introductory computer graphics course.

Dalton R. Hunkins

References

Hunkins, Dalton and Levine, David, "Additional Rich Resources for Computer Graphics Educators", *Computers and Graphics,* 2002, Vol. 26, No. 4, pp. 609-614.

Hunkins, Dalton and Levine, David, "Rich Resources for Computer Graphics Education", SIGGRAPH 2001 Educators Forum, Los Angeles, June 2001, pp. 45-48.

Appendix A: Annotated Table of Contents

Chapter 3: Transformations and Animation

Tutorial: 2D Transformations

By interacting with this tutorial, you can see how a 2D object is affected with the application of two-dimensional transformations while also viewing the current transformation matrix that creates the effect.

Example: Many Instances

We saw with the example Viewports how we can use different viewports to create several instances of a geometry. The use of transformations gives another way of creating multiple static instances of a geometry. The example illustrates this point.

Example: Morphing Using Idle Function

Animations are of great interest. We use the basic OpenGl transformations to animate the size, orientation and/or location of geometries. In this example we use transformations to change the shape of an object. We also animate the color of the object. In creating the animation, we demonstrate how to the use an idle function that we register with glut. In particular, our registered idle function calls our display method with a change between calls to the parameters that control the transformations on the object and the color applied to the object.

Example: Morphing Using Timer Function

The animation is the same as the one demonstrated in the previous example. Except in this example we create the animation through the use of a timer function that we register with glut. The advantage in using a timer function versus an idle function allows us to make the speed of the animation platform independent by setting the frame rate of the animation.

Chapter 4: Menu, Keyboard and Mouse Interaction

Example: User Interaction through GLUT

We see with this example how to create interactive methods using glut. These include the use of the mouse for picking an object within the world and dragging the object. We also see how to create a popup menu and define keyboard handlers.

Chapter 5: Texture Mappings

Tutorial: Texture Coordinates

The tutorial guides you through understanding what is needed to accomplish the application of a texture (bitmap) to a geometry in two dimensions.

Tutorial: Texture Transformations

The tutorial follows the one on texture coordinates. It is suggested that you do the Texture Coordinate tutorial before doing this one.

Example: Image Map

The example has two image maps where either of the two can be applied to different geometries. The geometries consist of squares, a hexagon and an octagon. The application of a map to a square is straight forward. But the calculations of the texture coordinates corresponding to the vertices of the hexagon and the octagon are a bit more complicated.

Chapter 6: A Little Game Programming

Example: Game with Stationary Targets

This example illustrates a simple third person shooting game. The game consists of a rotating tank that fires projectiles and five stationary targets. Discussion is given here on how we decide when a projectile will hit a target. This game provides one of the simplest cases of collision detection -- a moving object along a straight line and a stationary target.

Example: Game with Moving Target
This is the first in a series of two in which a moving target is added to our shooting game with stationary targets. Our method of collision detection used with stationary targets can be used with the moving target if the moving target stops when the projectile is fired. Guess what happens in this version of the game.

Example: Shooting Game (final version)
Collision detection in this version of the game is with both objects moving. The calculations are still relatively simple since the two motion paths are straight lines. Note that in a real situation, collision detection requires knowing not only the motion paths but also the velocities of each object. With this information we compute whether the two objects will arrive at the collision point at the same time. Time in our synthetic world is measured by frames and, in turn, our question is will the two moving objects arrive at the collision point in the same frame.

Chapter 7: Virtual Camera

Tutorial: Virtual Camera
A key concept in computer graphics is to model a world and then "take a picture" of the world through a (virtual) camera. We first establish the camera in OpenGL by setting its location, the direction it points and its orientation, also called its roll. Next, we establish the type of projection to be applied, either orthographic or perspective. The tutorial, in turn, is designed to help you understand this fundamental concept of viewing and how it is accomplished in OpenGL.

Tutorial: Camera Motion
The standard camera movements include orbit, pan, truck and dolly. By interacting with this tutorial, you can see the effect of each of these camera moves on the projected image.

Tutorial: The Viewing Transformation Matrix
The tutorial shows the computations used by OpenGL in determining the viewing transformation matrix. This matrix is created by OpenGL when a call is made to gluLookAt.

Chapter 8: Lighting

Tutorial: Ambient Reflection
The tutorial focuses on the determination of color using the Phong Illumination Model. In particular, you control the color of an ambient light as it illuminates four swatches of material (cyan, yellow, magenta, and gray). The color computations for any one of the materials you choose are shown in one sub window while the visual effect of the computation is displayed in another sub window.

Tutorial: Diffuse Reflection
The tutorial helps you understand the lighting effects of omni and directional lights on a simple scene consisting of two planes meeting at a corner. Further, the effect involves diffuse reflectivity and is determined by the orientation of surface normals to the lights. You can change the lighting effects by manipulating the normals.

Tutorial: Specular Reflection
The tutorial demonstrates specular reflectivity as created by a spot light illuminating a sphere. Also, you can change the cutoff angle and the exponent of the light and see the effect of the changes by viewing the illumination of the light on the background.

Example: Lights
The example illustrates the use of four lights, ambient, omni, directional and spot, to illuminate our world.

Chapter 9: Miscellaneous

Tutorial: Bezier Spline

A spline consists of curves placed end-to-end. We consider in this tutorial Bezier curves that are part of OpenGL. We also show how the splines obtained from these curves can be used as motion paths, to obtain a surface through extrusion and to obtain a surface of revolution.

Tutorial: A Projected Shadow

The tutorial demonstrates how a shadow can be cast onto a plane by computing a projection matrix and applying it to the 3D geometry. We also set the drawing color to one appropriate for the shadow and apply it to the projected geometry.

Tutorial: Ray Traced Shadow

The tutorial applies the method used in a Ray Trace render for creating shadows. The demonstration shows the calculations for "casting" a shadow onto a plane by one of three types of objects -- a sphere, a hollow cylinder (one without end caps) and a solid cylinder (one with caps).

Example: Two-Sided Material

OpenGL provides the means to assign different materials to two sides, the outside and the inside, of the geometry. In this example, you can cut away the geometry, which is of course hollow, and see both the inside and outside of the geometry. You create the cut-away by moving the near plane of the virtual camera.

Example: Inside a Sphere

Generally the background of our synthetic world is static. If we want the background to change as we animate the camera, then we can place the background on the inside of a sphere and have the sphere surround the geometry and the camera, as done in this example.

Example: Transparency

We have seen with the tutorial Texture Transformations how to make a portion of an image map "invisible." We do this by setting the alpha component of the texel's color to zero and apply alpha bending. This can also be accomplished when using a base color or material color for an object. The example also illustrates how to use the GLUI library in creating an interface for user input.

Example: Texture Methods

We have seen in chapter 5 two of the OpenGL methods, namely GL_REPLACE and GL_DECAL, for applying a image map to a geometry. A third method GL_MODULATE is demonstrated in the example. You can also compare the effect created with this method in contrast to GL_DECAL. Also, part of the user interface is created with the GLUI library. Note that this example is located here versus chapter 5 since the effect of modulate is best appreciated when lighting is present.

Example: Model Viewer

The tutorial Camera Animation is a model viewer but with only one model available. The example illustrates how to use the GLUI library to open a file browser and select a model to be viewed. Reading the file and rendering its content is part of the Utils class ObjectModel. You can use this class and any of the others in Utils when you construct your own programs.

Appendix B: Sample Scripts

Modifying Hello World

The source code for Hello World, namely HelloWorld.cpp, resides in the folder ../ExampleSourceCode/Chapter 0/.

- *Open the program HelloWorld.cpp in your IDE.*

As mentioned in the introduction, the IDE used in developing the tutorials is Bloodshed's Dev-C++ with the Cygwin C++ compiler. However, it does not matter which one you use as long as it is setup to compile programs that make calls to OpenGL and GLUT. The particulars on setting up a specific IDE will not be discussed here since there are many differences between them.

There are a number of changes that we can make to Hello World, each producing a different visual affect. In addition to changing the parameter passed to glLineWidth, try the following.

Discovery Activity: What colors do we obtain?

- *Using only 0's and 1's, vary the parameters passed to glColor3f. Try each of the eight possible combinations (1, 1, 1), (0, 1, 1), etc. Compile and run your modified program. Note that color mixing is covered with the tutorial in chapter 2.*

Discovery Activity: Assigning colors to each vertex

- *Place a copy of the call to glColor3f before each of the four calls to glVertex2d. Assign different color values to the four calls to glColor3f. Compile and run the modified program and observe how these changes affect the rendered image.*

Discovery Activity: What happens if we use a coordinate larger than 1.0 or smaller than 0.0?

- *Change one or more of the vertices so that at least one of its coordinates (x or y or both) is bigger than 1.0 and/or less than 0.0. How does this change the rendered image? Keep in mind the boundaries of the projection window are defined in the call to gluOrtho2D,.*

Discovery Activity: Does the call to glFlush really matter?

- *Comment out the call to glFlush and compile and run the modified program. Is the call to glFlush necessary? Why?*

Exploring with the Orthographic Projection Tutorial

Run the Orthographic Projection tutorial that is in chapter 1.

Discovery Activity: Setting the Projection Window

We are going to change the boundaries of the projection window so as to select certain portions of the world for rendering. You accomplish this by changing the boundary parameters appearing in the parameter input window on the bottom-left side. Also, the technical content window appearing on the bottom-right side shows the coordinates of the rectangles.

- *Set the boundaries of the projection window so that only the red rectangle appears in the actual display window with a black border around it.*

Obviously you can carry out this activity by simply changing the boundaries while watching the projection window in the road-map window. But keep in mind that this visual is not available when writing your graphics programs; you will be setting the projection window boundaries based on the coordinates of the geometries within your world. Uncheck the box "Show Projection Window" in the selection panel and carry out the next activity.

- With the projection window not visible in the road-map window, s*et the boundaries of the projection window so that only the red and green rectangles appear in the actual display window with a black border around them.*

Did you pick the left boundary smaller than -30 (e.g. -31) and the right boundary bigger than 15 (e.g., 16)? Also did you pick the bottom boundary smaller than -40 (e.g., -41) and the top boundary larger than 15 (e.g., 16)? How can we determine these boundary values from the coordinates of the rectangles?

- Looking at the coordinates of the rectangles, can you pick the boundaries of the projection window so that the red, green, blue and yellow rectangles are displayed AND no portion of any other rectangle? Why or why not?

Discovery Activity: Setting the Viewport

Observe that the shape of the cyan rectangle is tall and skinny. Let us see how it is displayed when we make it the only object in the projection window.

- With the projection window not visible in the road-map window, s*et the boundaries of the projection window so that only the cyan rectangle appears in the actual display window with a black border around them.*

Depending on the amount of border you leave around the rectangle, the rectangle appears almost as a square when rendered to the display window. Remember that the projection window is mapped (rendered) to the entire viewport. So, if the projection window is tall and skinny and the viewport is square, the contents of the projection window will be stretched and distorted. In order to avoid the distortion, we have to set the dimensions of the viewport so that it has the same aspect ratio as the projection window. We define the ***aspect ratio*** of a rectangle as the quotient

width of rectangle / height of rectangle

A call to glViewport is shown in the parameter input window. The changeable values displayed here are percentages of the width and height; as mentioned, this is a convenient way of setting the boundaries of the viewport.

- Keeping the projection window set so as to contain only the cyan rectangle and leaving the left and bottom boundaries of the viewport at 0, change the dimensions of the viewport so that its aspect ratio is the same as that of the projection window.

Did you observe that since the desired aspect ratio is less than one, we need only decrease the width of the viewport to achieve that ratio? Also, the rendered image appears on the left side of the actual display window.

- Keeping the aspect ratio you set, change a boundary (or boundaries) of the viewport so that the displayed image of the cyan rectangle appears centered in the actual display window.

Did you change only the left boundary of the viewport to achieve the desired affect?

We close this section with the following suggestion.

Use a piece of graph paper when designing your virtual world and label the coordinates of all the vertices within your world.

Exploring with the Color Models Tutorial

Run the Color Models tutorial that is in chapter 2.

Discovery Activity: What is Hue?

- *Click on the second value of rgb[], which sets the green value in RGB, and drag the mouse upward, changing the parameter value from 0.0 to 1.0. Observe the changes in the color models with changes to this green value.*

Observe how the color changes from red to yellow. Also note how the ball moves in the RGB model and the HSV model. You will also see the value for hue changing from 0.0 to 60.0.

- *Click on the first value in rgb[], which sets the red value, and drag the mouse downward, changing the value from 1.0 to 0.0.*

Observe that the ball moves along the edge of the hexcone base and the edge of the cube that connects yellow to green. Also the hue parameter changes value from 60.0 to 120.0.

- *Use the mouse to change the blue value from 0.0 to 1.0*

What color changes take place and how does the value for hue change? You should observe that the color changes from green to cyan and hue changes from 120.0 to 180.0.

- *Use the mouse to change the green value back to 0.0.*
- *After which, increase the red value to 1.0.*
- *Last, decrease the blue value to 0.0, taking us back to full red.*

Note that the ball continues to move along the base of the hexcone in a counterclockwise fashion and the value of hue continues to increase through the values 240, 300 and then back to 0. Also observe how the ball moves in the RGB cube, namely along the edges connecting the primary (red, green and blue) and secondary (yellow, cyan and magenta) colors.

Hue is generally what people refer to when talking about color; it is similar to the colors of the rainbow. Hue in the HSV model is represented by the angle in degrees measured counterclockwise around the center of the base of the hexcone. The range of hues is [0, 360) with red = 0, yellow = 60, green = 120, etc. Let us confirm our observations with the following.

- *Click on the hue value and drag the mouse upward, changing the value from 0.0 through the values 60.0, 120.0, 180.0, 240.0, 300.0 and then back to 0.0.*

Yes, the ball moves along the edges of the base of the hexcone in the HSV model and along the edges of the RGB cube connecting the primary and secondary colors.

We have been observing color thus far at its full purity and brightness. The values in the HSV model that control these properties are respectfully saturation and value. The two can both be set with numbers in the range [0.0, 1.0].

Discovery Activity: What is Saturation?

- *Click on the saturation component for the array hsv[] and drag the mouse up and down, changing it between the numbers 0.0 to 1.0.*

Observe that when hue = 0, the color goes between red and white through variations of pink. Thus, saturation measures the purity of the color where maximum purity is when saturation = 1.0. When saturation = 0.0 (and value = 1.0), the color is white. Observe also the changes that take place in the RGB model. As the color red decreases in purity, the green and blue components increase equally to 1.0 resulting in the color white and the ball at the vertex (1.0, 1.0, 1.0). Similar changes occur for any of the primary and secondary colors.

- *Set red and green to 1.0 and blue to 0.0 in the RGB model and, in turn, make the color yellow. Now change saturation from 1.0 to 0.0.*

Observe how the color changes to a pale shade of yellow and finally to white. Note also the blue component in the RGB model increases to 1.0.

Discovery Activity: What is Value?

Value, sometimes called brightness, measures the total energy of the color (light) where maximum brightness occurs when value = 1.0. When value is 0.0, the color is black.

- *Set saturation to 0.0 and value to 1.0 in the HSV model. (Or equivalently, set each of the parameters in the RGB model to 1.0.) Now change the value component of hsv[] from 1.0 to 0.0.*

Observe that the ball in each of the models moves along the gray-scale from white to black through shades of gray. In the RGB model, a shade of gray is when the parameters (red, green and blue) are set to the same number.

In summary, saturation in the HSV model measures the relative distance of a point to its axis (gray scale) while value measures the relative height of the point from the apex of the hexcone. We can also think of saturation and value in the RGB model, although it is not as intuitive as in the HSV model. Since saturation is the purity of the color and corresponds to the difference between the color and white, we can reduce the saturation in the RGB model by moving the color from its current location towards the gray scale. We can also change a color's value in the RGB model by moving along the line going from the color's current position to black. For example, to reduce the value of magenta, which is at the point (1.0, 0.0, 1.0), the point moves along the diagonal line in the red-blue face of the cube from the magenta vertex to the black vertex (0.0, 0.0, 0.0). What happens to saturation when the color reaches black? (Hint: any point on the gray scale has zero saturation). Try it!