Foundations of 3D Graphics Programming
Jim X. Chen
Edward J. Wegman

Foundations of 3D Graphics Programming
Using JOGL and Java3D

With 139 Figures
Preface

OpenGL, which has been bound in C, is a seasoned graphics library for scientists and engineers. As we know, Java is a rapidly growing language becoming the de facto standard of Computer Science learning and application development platform as many undergraduate computer science programs are adopting Java in place of C/C++. Released by Sun Microsystems in June 2003, the recent OpenGL binding with Java, JOGL, provides students, scientists, and engineers a new venue of graphics learning, research, and applications.

Overview

This book aims to be a shortcut to graphics theory and programming in JOGL. Specifically, it covers OpenGL programming in Java, using JOGL, along with concise computer graphics theories. It covers all graphics basics and several advanced topics without including some implementation details that are not necessary in graphics applications. It also covers some basic concepts in Java programming for C/C++ programmers. It is designed as a textbook for students who know programming basics already. It is an excellent shortcut to learn 3D graphics for scientists and engineers who understand Java programming. It is also a good reference for C/C++ graphics
programmers to learn Java and JOGL. This book is a companion to Guide to Graphics Software Tools (Springer-Verlag, New York, ISBN 0-387-95049-4), which covers a smaller graphics area with similar examples in C but has a comprehensive list of graphics software tools.

Organization and Features

This book concisely introduces graphics theory and programming in Java with JOGL. A top-down approach is used to lead the audience into programming and applications up front. The theory provides a high-level understanding of all basic graphics principles without some detailed low-level implementations. The emphasis is on understanding graphics and using JOGL instead of implementing a graphics system. The contents of the book are integrated with the sample programs, which are specifically designed for learning and accompany this book. To keep the book’s conciseness and clarity as a high priority, the sample programs are not production-quality code in some perspectives. For example, error handling, GUI, controls, and exiting are mostly simplified or omitted.

Chapter 1 introduces OpenGL, Java, JOGL, and basic graphics concepts including object, model, image, frame buffer, scan-conversion, clipping, and antialiasing. Chapter 2 discusses transformation theory, viewing theory, and OpenGL programming in detail. 3D models, hidden-surface removal, and collision detection are also covered. Chapter 3 overviews color in hardware, eye characteristics, gamma correction, interpolation, OpenGL lighting, and surface shading models. The emphasis is on OpenGL lighting. Chapter 4 surveys OpenGL blending, image rendering, and texture mapping. Chapter 5 introduces solid models, curves, and curved surfaces. Chapter 6 discusses scene graph and Java3D programming with concise examples. Chapter 7 wraps up basic computer graphics principles and programming with some advanced concepts and methods.

Web Resources

JOGL and Java3D sample programs (their sources and executables) are available online. The following Web address contains all the updates and additional
information, including setting up the OpenGL programming environment and accompanying Microsoft PowerPoint course notes for learners and instructors:

http://cs.gmu.edu/~jchen/graphics/jogl/

**Audience**

The book is intended for a very wide range of readers, including scientists in different disciplines, undergraduates in Computer Science, and Ph.D. students and advanced researchers who are interested in learning and using computer graphics on Java and JOGL platform.

Chapters 1 through 4 are suitable for a one-semester graphics course or self-learning. These chapters should be covered in order. Prerequisites are preliminary programming skills and basic knowledge of linear algebra and trigonometry. Chapters 5 and 6 are independent introductions suitable for additional advanced graphics courses.

**Acknowledgments**

As a class project in CS 652 at George Mason University, Danny Han initially coded some examples for this book. We acknowledge the anonymous reviewers and the whole production team at Springer. Their precious comments, editings, and help have significantly improved the quality and value of the book.

Jim X. Chen and Edward J. Wegman
May 2006
Contents

Chapter 1

Introduction  1

1.1 Graphics Models and Libraries  - - - - - - - - - - - - - - 1

1.2 OpenGL Programming in Java: JOGL  - - - - - - - - - - - -  2
  1.2.1 Setting Up Working Environment  2
  1.2.2 Drawing a Point  7
  1.2.3 Drawing Randomly Generated Points  9

1.3 Frame Buffer, Scan-conversion, and Clipping  - - - - - 11
  1.3.1 Scan-converting Lines  12
  1.3.2 Scan-converting Curves, Triangles, and Polygons  18
  1.3.3 Scan-converting Characters  20
  1.3.4 Clipping  21

1.4 Attributes and Antialiasing  - - - - - - - - - - - - - -  22
  1.4.1 Area Sampling  22
  1.4.2 Antialiasing a Line with Weighted Area Sampling  24

1.5 Double-buffering for Animation  - - - - - - - - - - - - -  28
Chapter 3

3.1 RGB Mode and Index Mode

3.1.1 RGB Mode and Index Mode

3.1.2 Eye Characteristics and Gamma Correction

3.2 Color Interpolation

3.3 Lighting

3.3.1 Lighting Components

3.3.2 OpenGL Lighting Model

3.4 Visible-Surface Shading

3.4.1 Back-Face Culling

3.4.2 Polygon Shading Models

3.4.3 Ray Tracing and Radiosity

3.5 Review Questions

3.6 Programming Assignments

Chapter 4

Blending and Texture Mapping

4.1 Blending

4.1.1 OpenGL Blending Factors

4.1.2 Transparency and Hidden-Surface Removal

4.1.3 Antialiasing

4.1.4 Fog

4.2 Images

4.3 Texture Mapping

4.3.1 Pixel and Texel Relations

4.3.2 Texture Objects

4.3.3 Texture Coordinates

4.3.4 Levels of Detail in Texture Mapping

4.4 Review Questions

4.5 Programming Assignments
Chapter 5

Curved Models  191

5.1 Introduction  - - - - - - - - - - - - - - - - - - - - - - - 191

5.2 Quadratic Surfaces  - - - - - - - - - - - - - - - - - - - - - - - - - - 192

5.2.1 Sphere  192
5.2.2 Ellipsoid  193
5.2.3 Cone  194
5.2.4 Cylinder  194
5.2.5 Texture Mapping on GLU Models  195

5.3 Tori, Polyhedra, and Teapots in GLUT  - - - - - - - - - - - - - - - - - 198

5.3.1 Tori  198
5.3.2 Polyhedra  198
5.3.3 Teapots  199

5.4 Cubic Curves  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 202

5.4.1 Continuity Conditions  203
5.4.2 Hermite Curves  205
5.4.3 Bezier Curves  208
5.4.4 Natural Splines  213
5.4.5 B-splines  214
5.4.6 Non-uniform B-splines  217
5.4.7 NURBS  218

5.5 Bi-cubic Surfaces  - - - - - - - - - - - - - - - - - - - - - - - - - - - - 219

5.5.1 Hermite Surfaces  219
5.5.2 Bezier Surfaces  221
5.5.3 B-spline Surfaces  225

5.6 Review Questions  - - - - - - - - - - - - - - - - - - - - - - - - - - - - 225

5.7 Programming Assignments  - - - - - - - - - - - - - - - - - - - - - - - 226

Chapter 6

Programming in Java3D  227

6.1 Introduction  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 227

6.2 Scene Graph  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 227
Chapter 7

Advanced Topics 257

7.1 Introduction 257

7.2 Graphics Libraries 258

7.3 Visualization 258

7.3.1 Interactive Visualization and Computational Steering 258

7.3.2 Data Visualization: Dimensions and Data Types 259

7.3.3 Parallel Coordinates 261

7.4 Modeling and Rendering 262

7.4.1 Sweep Representations 263

7.4.2 Instances 263

7.4.3 Constructive Solid Geometry 263

7.4.4 Procedural Models 263

7.4.5 Fractals 264

7.4.6 Particle Systems 264

7.4.7 Image-based Modeling and Rendering 265
### 7.5 Animation and Simulation

- 7.5.1 Physics-based Modeling and Simulation 268
- 7.5.2 Real-Time Animation and Simulation: A Spider Web 270
- 7.5.3 The Efficiency of Modeling and Simulation 273

### 7.6 Virtual Reality

- 7.6.1 Hardware and Software 275
- 7.6.2 Non-immersive Systems 275
- 7.6.3 Basic VR System Properties 276
- 7.6.4 VR Tools 276
- 7.6.5 VR Simulation Tools 277
- 7.6.6 Basic Functions in VR Tool 278
- 7.6.7 Characteristics of VR 278

### 7.7 Graphics on the Internet: Web3D

- 7.7.1 Virtual Reality Modeling Language (VRML) 280
- 7.7.2 X3D 280
- 7.7.3 Java3D 280

### 7.8 3D File Formats

- 7.8.1 3D File Formats 281
- 7.8.2 3D Programming Tool Libraries 282
- 7.8.3 3D Authoring Tools 282
- 7.8.4 3D File Format Converters 282
- 7.8.5 Built-in and Plug-in VRML Exporters 283
- 7.8.6 Independent 3D File Format Converters 283

### 7.9 3D Graphics Software Tools

- 7.9.1 Low-Level Graphics Libraries 284
- 7.9.2 Visualization 284
- 7.9.3 Modeling and Rendering 285
- 7.9.4 Animation and Simulation 287
- 7.9.5 Virtual Reality 288
- 7.9.6 Web3D 288
- 7.9.7 3D File Format Converters 289
1 Introduction

Chapter Objectives:

- Introduce basic graphics concepts — object, model, image, graphics library, framebuffer, scan-conversion, clipping, and antialiasing
- Set up Java, JOGL programming environments
- Understand simple JOGL programs

1.1 Graphics Models and Libraries

A graphics display is a drawing area composed of an array of fine points called pixels. At the heart of a graphics system there is a magic pen, which can move at lightning speed to a specific pixel and draw the pixel with a specific color — a red, green, and blue (RGB) vector value. This pen can be controlled directly by hand through an input device (mouse or keyboard) like a simple paintbrush. In this case, we can draw whatever we imagine, but it takes a real artist to come up with a good painting. Computer graphics, however, is about using this pen automatically through programming.

A real or imaginary object is represented in a computer as a model and is displayed as an image. A model is an abstract description of the object’s shape (vertices) and attributes (colors), which can be used to find all the points and their colors on the object corresponding to the pixels in the drawing area. Given a model, the application program will control the pen through a graphics library to generate the corresponding image. An image is simply a 2D array of pixels.

A graphics library provides a set of graphics commands or functions. These commands can be bound in C, C++, Java, or other programming languages on different platforms. Graphics commands can specify primitive 2D and 3D geometric
models to be digitized and displayed. Here *primitive* means that only certain simple shapes (such as points, lines, and polygons) can be accepted by a graphics library. To draw a complex shape, we need an application program to assemble or construct it by displaying pieces of simple shapes (primitives). We have the magic pen that draws a pixel. If we can draw a pixel, we can draw a line, a polygon, a curve, a block, a building, an airplane, and so forth. A general application program can be included into a graphics library as a command to draw a complex shape. Because our pen is magically fast, we can draw a complex object, clear the drawing area, draw the object at a slightly different location or shape, and repeat the above processes — the object is now animated.

*OpenGL* is a graphics library that we will integrate with the *Java programming language* to introduce graphics theory, programming, and applications. When we introduce program examples, we will succinctly discuss Java-specific concepts and programming as well for C/C++ programmers.

### 1.2 OpenGL Programming in Java: JOGL

OpenGL is the most widely used graphics library (GL) or application programming interface (API), and is supported across all popular desktop and workstation platforms, ensuring wide application deployment. *JOGL* implements Java bindings for OpenGL. It provides hardware-supported 3D graphics to applications written in Java. It is part of a suite of open-source technologies initiated by the Game Technology Group at Sun Microsystems. JOGL provides full access to OpenGL functions and integrates with the AWT and Swing widget sets.

First, let’s spend some time to set up our working environment, compile J1_0_Point.java, and run the program. The following file contains links to all the example programs in this book and detailed information for setting up working environments on different platforms for the most recent version:

http://cs.gmu.edu/~jchen/graphics/setup.html

#### 1.2.1 Setting Up Working Environment

JOGL provides full access to the APIs in the OpenGL 1.4 specification as well as nearly all vendor extensions. To install and run JOGL, we need to install Java
Development Kit. In addition, a Java IDE is also preferred to help coding. The following steps will guide you through installing Java, JOGL, and Eclipse or JBuilder IDE.

1. Installing Java Development Kit 1.4 or Above

Java Development Kit (JDK) contains a compiler, interpreter, and debugger. If you have not installed JDK, it is freely available from Sun Microsystems. You can download the latest version from the download section at http://java.sun.com. Make sure you download the JDK (or SDK) not the JRE (runtime environment) that matches the platform you use. For example, version 1.5.0 can be downloaded from Java2 Standard Edition (J2SE) at http://java.sun.com/j2se/1.5.0/download.jsp. After downloading the JDK, you can run the installation executable file. During the installation, you will be asked the directory “Install to:". You need to put it somewhere you know. For example: “C:\j2sdk1.5.0\”.

2. Installing JOGL

The first step required is to obtain the binaries that you will need in order to compile and run your applications. These pre-compiled binaries can be obtained from the project Web site (https://jogl.dev.java.net/) Precompiled binaries and documentation. Go to Release Builds 2005 and download “jogl.jar”, and then download the binaries that match the platform you use. For Windows platform, for example, it is named “jogl-natives-win32.jar”. After downloading “jogl-natives-win32.jar”, you should extract “jogl.dll” and jogl_cg.dll” from it by the following command:

```
"C:\j2sdk1.5.0\bin\jar" -xvf jogl-natives-win32.jar
```

After that, you can put the three files (jogl.jar, jogl.dll, and jogl_cg.dll) in the directory with the Java (JOGL) examples and compile all them on the command line in the current directory with:

```
"C:\j2sdk1.5.0\bin\javac" -classpath jogl.jar *.java
```

After that, you can run the sample program with (the command in one line):

```
"C:\j2sdk1.5.0\bin\java" -classpath .;jogl.jar;
-Djava.library.path=. J1_0_Point
```
That is, you need to place the “jogl.jar” file in the CLASSPATH of your build environment in order to be able to compile an application with JOGL and run, and place “jogl.dll” and “jogl_cg.dll” in the directory listed in the “java.library.path” environment variable during execution. Java loads the native libraries (such as the dll file for Windows) from the directories listed in the “java.library.path” environment variable. For Windows, placing the dll files under “C:\WINDOWS\system32\” directory works. This approach gets you up running quickly without worrying about the “java.library.path” setting.

3. Installing a Java IDE (Eclipse, jGRASP, or JBuilder)

Installing a Java IDE (Integrated Development Environment) is optional. Without an IDE, you can edit Java program files using any text editor, compile and run Java programs using the commands we introduced above after downloading JOGL.

Java IDEs such as Eclipse, JBuilder, or jGRASP are development environments that make Java programming much faster and easier. If you use Eclipse, you can put “jogl.jar” in “C:\j2re1.5.0\lib\ext\” directory in the Java runtime environment.

You can download from [http://eclipse.org](http://eclipse.org) the latest version of Eclipse that matches the platform you use. Expand it into the folder where you would like Eclipse to run from, (e.g., “C:\eclipse\”). There is no installation to run. To remove Eclipse you simply delete the directory, because Eclipse does not alter the system registry.

If you use jGRASP, in the project under “compiler->setting for workspace->PATH”, you can add the directory of the *.dll files to the system PATH window, and add “jogl.jar” file with full path to the CLASSPATH window.

As an alternative, you can download a free version of JBuilder from [http://www.borland.com/jbuilder/](http://www.borland.com/jbuilder/). JBuilder comes with its own JDK. If you use JBuilder as the IDE and want to use your downloaded JDK, you need to start JBuilder, go to "Tools->Configure JDKs", and click "Change" to change the "JDK home path:" to where you install your JDK. For example, “C:\j2sdk1.5.0\”. Also, under "Tools->Configure JDKs", you can click “Add” to add “jogl.jar” from wherever you save it to the JBuilder environment.

4. Creating a Sample Program in Eclipse

As an example, here we introduce using Eclipse. After downloading it, you can run it to start programming. Now in Eclipse you click on “File->New->Project” to create a new Java Project at a name you prefer. Then, you click on
“File->New->Class” to create a new class with name: “J1_0_Point”. After that, you can copy the following code into the space, and click on “Run->Run As->Java Application” to start compiling and running. You should see a window with a very tiny red pixel at the center. In the future, you can continue creating new classes, as we introduce each example as a new class.

/* draw a point */
/* Java’s supplied classes are “imported”. Here the awt (Abstract Windowing Toolkit) is imported to provide “Frame” class, which includes windowing functions */
import java.awt.*;

// JOGL: OpenGL functions
import net.java.games.jogl.*;

/* Java class definition: “extends” means “inherits”. So J1_0_Point is a subclass of Frame, and it inherits Frame’s variables and methods. “implements” means GLEventListener is an interface, which only defines methods (init(), reshape(), display(), and displaychanged()) without implementation. These methods are actually callback functions handling events. J1_0_Point will implement GLEventListener’s methods and use them for different events. */
public class J1_0_Point extends Frame implements GLEventListener {

    static int HEIGHT = 400, WIDTH = 400;
    static GL gl; //interface to OpenGL
    static GLCanvas canvas; // drawable in a frame
    GLCapabilities capabilities; // OpenGL capabilities

    public J1_0_Point() { // constructor

        //1. specify a drawable: canvas
        capabilities = new GLCapabilities();
        canvas = GLDrawableFactory.getFactory().createGLCanvas(capabilities);

        //2. listen to the events related to canvas: reshape
        canvas.addGLEventListener(this);

        //3. add the canvas to fill the Frame container
        add(canvas, BorderLayout.CENTER);
        /* In Java, a method belongs to a class object. Here the method “add” belongs to J1_0_Point’s instantiation, which is frame in “main” function. */
It is equivalent to use “this.add(canvas, ...)” */

//4. interface to OpenGL functions
gl = canvas.getGL();

public static void main(String[] args) {
    J1_0_Point frame = new J1_0_Point();

    //5. set the size of the frame and make it visible
    frame.setSize(WIDTH, HEIGHT);
    frame.setVisible(true);
}

// Called once for OpenGL initialization
public void init(GLDrawable drawable) {

    //6. specify a drawing color: red
    gl.glColor3f(1.0f, 0.0f, 0.0f);
}

// Called for handling reshaped drawing area
public void reshape(GLDrawable drawable, int x, int y,
                     int width, int height) {

    //7. specify the drawing area (frame) coordinates
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glOrtho(0, width, 0, height, -1.0, 1.0);
}

// Called for OpenGL rendering every reshape
public void display(GLDrawable drawable) {

    //8. specify to draw a point
    gl.glBegin(GL.GL_POINTS);
    gl.glVertex2i(WIDTH/2, HEIGHT/2);
    gl.glEnd();
}

// called if display mode or device are changed
public void displayChanged(GLDrawable drawable,
                           boolean modeChanged, boolean deviceChanged) {
}
1.2.2 Drawing a Point

The above J1_0_Point.java is a Java application that draws a red point using JOGL. If you are a C/C++ programmer, you should read all the comments in the sample program carefully, because they include explanations about Java-specific terminologies and coding. Our future examples are built on top of this one. Here we explain in detail. The program is complex to us at this point of time. We only need to understand the following:

1. Class GLCanvas is an Abstract Window Toolkit (AWT) component that provides OpenGL rendering support. Therefore, the GLCanvas object, canvas, corresponds to the drawing area that will appear in the Frame object frame, which corresponds to the display window. Here object means an instance of a class in object-oriented programming, not a 3D object. In the future, we omit using a class name and underline its object name in our discussion. In many cases, object names are lowerscases of the corresponding class names to facilitate understanding.

2. An event is a user input or a system state change, which is queued with other events to be handled. Event handling is to register an object to act as a listener for a particular type of event on a particular component. Here frame is a listener for the GL events on canvas. When a specific event happens, it sends canvas to the corresponding event handling method and invokes the method. GLEventListener has four event-handling methods:

   - `init()` is called immediately after the OpenGL context is initialized for the first time, which is a system event. It can be used to perform one-time OpenGL initialization;

   - `reshape()` is called if canvas has been resized, which happens when the user changes the size of the window. The listener also passes the drawable canvas and the display area’s lower-left corner (x, y) and size (width, height) to the method. At this time, (x, y) is always (0, 0), and the canvas’ size is the same as the display window’s frame. The client can update the coordinates of the display corresponding to the resized window appropriately. reshape() is called at least once when program starts. Whenever reshape() is called, display() is called as well;

   - `display()` is called to initiate OpenGL rendering when program starts. It is called afterwards when reshape event happens;
• `displayChanged()` is called when the display mode or the display device has been changed. Currently we do not use this event handler.

3. `canvas` is added to `frame` to cover the whole display area. `canvas` will reshape with `frame`.

4. `gl` is an interface handle to OpenGL methods. All OpenGL commands are prefixed with “`gl`” as well, so you will see OpenGL method like `gl.glColor()`. When we explain the OpenGL command, we often omit the interface handle.

5. Here we set the physical size of `frame` and make its contents visible. Here the physical size corresponds to the number of pixels in `x` and `y` direction. The actual physical size also depends on the resolution of the display, which is measured in number of pixels per inch. At this point, the window frame appears. Depending on the JOGL version, the physical size may include the boarders, which is a little larger than the visible area that is returned as `w` and `h` in `reshape()`.

6. The foreground drawing color is specified as a vector (red, green, blue). Here (1, 0, 0) represents a red color.

7. These methods specify the logical coordinates. For example, if we use the command `glOrtho(0, width, 0, height, −1.0, 1.0)`, then the coordinates in `frame` (or `canvas`) will be $0 \leq x \leq width$ from the left side to the right side of the window, $0 \leq y \leq height$ from the bottom side to the top side of the window, and $−1 \leq z \leq 1$ in the direction perpendicular to the window. The $z$ direction is ignored in 2D applications. It is a coincidence that the logical coordinates correspond to the physical (pixel) coordinates, because `width` and `height` are initially from `frame`’s `WIDTH` and `HEIGHT`. We can specify `glOrtho(0, 100*WIDTH, 0, 100*HEIGHT, −1.0, 1.0)` as well, then point $(WIDTH/2, HEIGHT/2)$ will appear at the lower-left corner of the `frame` instead of the center of the `frame`.

8. These methods draw a point at $(WIDTH/2, HEIGHT/2)$. The coordinates are logical coordinates not directly related to the canvas’ size. The `width` and `height` in `glOrtho()` are actual window size. It is the same as `WIDTH` and `HEIGHT` at the beginning, but if you reshape the window, they will be different, respectively. Therefore, if we reshape the window, the red point moves.

In summary, when Frame is instantiated, constructor `J1_0_Point()` will create a drawable canvas, add event listener to it, attach the display to it, and get a handle to `gl` methods from it. `reshape()` will set up the display’s logical coordinates in the window frame. `display()` will draw a point in the logical coordinates. When program starts,