
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) <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 website (<https://jogl.dev.java.net/>) [Precompiled binaries and documentation](#). Go to [Release Builds 2003](#) and make sure you download the binaries that match the platform you use. For Windows platform, for example, it is named [jogl-win32.zip](#). After extracting the binaries, you have “jogl.jar”, “jogl.dll”, and “jogl_cg.dll” files. If you go to [Release Builds 2004](#), after downloading “jogl.jar”, you should download “jogl-natives-win32.jar” and extract “jogl.dll” and [jogl_cg.dll](#) from it. You can put all them in the directory with the Java (JOGL) examples, and compile all them on command line in the current directory with:

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

After that, you can run a sample program with:

```
“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 source 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 the latest version of Eclipse from <http://eclipse.org> 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 formal installation to run. To remove Eclipse you simply delete the Eclipse 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/>. 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
Jl_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.
Jl_0_Point will implement GLEventListener's methods and use
them for different events.
*/
public class Jl_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

    public Jl_0_Point() { // constructor

        //1. specify a drawable: canvas
        GLCapabilities 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 Jl_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 lowercases 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 its coordinates and size (x, y, w, h) to the method. 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. At this point, the window frame appears. Depending on the JOGL version, the physical size may include the borders, 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 *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, since *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, *main()* will be called, then frame instantiation, *J1_0_Point()*, *setSize()*, *setVisible()*, *init()*, *reshape()*, and *display()*. *reshape()* and *display()* will be called again and again if the user changes the display area. You may not find it, but a red point appears in the window.

1.2.3 Drawing Randomly Generated Points

J1_1_Point extends *J1_0_Point*, so it inherits all the methods from *J1_0_Point* that are not private. We can reuse the constructor and some of the methods.

```
/* draw randomly generated points */
```

```
import net.java.games.jogl.*;
import java.awt.event.*;

//built on J1_0_Point class through inheritance
public class J1_1_Point extends J1_0_Point {

    static Animator animator; // drive display() in a loop

    public J1_1_Point() {

        //1. use super's constructor to initialize drawing
        // which is done automatically

        //2. add a listener for window closing
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                animator.stop(); // stop animation
                System.exit(0);
            }
        });
    }

    // Called one-time for OpenGL initialization
    public void init(GLDrawable drawable) {

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

```
//3. clear the background to black
gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
gl.glClear(GL.GL_COLOR_BUFFER_BIT);

//4. drive the display() in a loop
animator = new Animator(canvas);
animator.start(); // start animator thread
}

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

    //5. generate a random point
    double x = Math.random() * WIDTH;
    double y = Math.random() * HEIGHT;

    // specify to draw a point
    gl.glBegin(GL.GL_POINTS);
        gl.glVertex2d(x, y);
    gl.glEnd();
}

public static void main(String[] args) {
    J1_1_Point f = new J1_1_Point();

    f.setTitle("JOGL");
    f.setSize(WIDTH, HEIGHT);
    f.setVisible(true);
}
}
```

1. *J1_1_Point* is built on (extends) the previous class, so we can reuse its methods. The super class's constructor is automatically called to initialize drawing and event handling.
2. In order to avoid window hanging, we add a listener for window closing, and stop animation before exit. Animation ([animator](#)) will be discussed later here.
3. *glClearColor()* specifies the background color. OpenGL is a state machine, which means that if we specifies the color, unless we change it, it will always be the same. Therefore, whenever we call *glClear()*, the background will be black unless we call *glClearColor()* to set it differently.