Computer Graphics

Jim X. Chen, Ph.D.
Director of Computer Graphics Lab
George Mason University

Copyright @ 2005 by Jim X. Chen:
jchen@cs.gmu.edu
Reference

Objects and Models

• **Objectives**
  
  – Introduce basic graphics concepts
    
    • object, model, image, graphics library, frame buffer, scan-conversion, clipping, and anti-aliasing
  
  – Set up OpenGL programming environment
  
  – Understand simple OpenGL programs
Computer Graphics

- CG displays or animates real or imaginary objects from their computer-based models;
  - Modeling creating and modifying a model
- Image processing treats the inverse process: the analysis of images, pattern recognition, or the reconstruction of 2D or 3D objects from images.
Display

- A graphics *display* is a drawing area comprised of an array of fine points called *pixels* (picture elements).

- At the heart of a graphics system there is a *magic pen*,
  - move at lightning speed to a specific pixel
  - draw the pixel with a specific color — a red, green, and blue (RGB) vector value.
  - Computer graphics is about using this pen automatically through programming.
Object, Model, and Image

• 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 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.
Interactive

• User controls the creation, modification, and animation of the models through input devices (keyboard & mouse)
**Primitive and Graphics Library**

- A *graphics library* provides a set of graphics commands or functions.
  - bound in C, Java, or other programming languages on different platforms.
  - specify primitive 2D and 3D geometric models to be digitized and displayed.

- *Primitive models* or simply *primitives* stand for some simple shapes (such as points, lines, and polygons)

- *OpenGL* is a graphics library; *DirectX* includes a graphics library *Direct3D*
OPENGL DRAWING PRIMITIVES

```c
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(3.0, 3.0);
    glVertex2f(4.0, 1.5);
    glVertex2f(3.0, 0.0);
glEnd();
```
OpenGL Programming

• OpenGL is the most widely used graphics library (GL) or application programming interface (API)

• Compile and run Example *J1_0_Point.java*
  
  – To change the font size in Eclipse: Window->Preferences… General->Colors and Fonts->Basic->Text Font, then select.

  – Corresponding example in C: 1.1.point.c

• links to all the example programs, and setting up working environments on different platforms:
  
/* 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 javax.media.opengl.*;
import javax.media.opengl.awt.*;

/* 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

    …; // methods
}

Copyright @ 2005 by Jim X. Chen:
jchen@cs.gmu.edu
public class J1_0_Point extends Frame implements GLEventListener {

    ...

    public J1_0_Point() { // constructor

        //1. specify a drawable: canvas
        canvas = new GLCanvas();

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

        ...
    }

    ...
}
public J1_0_Point() { // constructor

  //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();
    ...
}

Copyright @ 2005 by Jim X. Chen:
jchen@cs.gmu.edu
public class J1_0_Point extends Frame implements GLEventListener {
    ...
    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);
    }
    ...
}
public class J1_0_Point extends Frame implements GLE ventListener {

...

// 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);
}

...

}
public class J1_0_Point extends Frame implements GLEventListener {
    ...

    // 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) {
    }
}
Double Buffers

• By default, we have two framebuffers, the front buffer and the back buffer.
• You can specify to draw into both front and back buffers if you are not generating an animation
  – `gl.glDrawBuffer(GL.GL_FRONT_AND_BACK);`
/* Example 1.1.point.c: draw randomly generated points */
#include <stdlib.h>
#include <GL/glut.h>

#define Height 400
#define Width 400

Points
void display(void)
{
    int x, y;

    //a. generate a random point
    x = rand() % Width;
    y = rand() % Height;

    //b. specify a drawing color: red
    glColor3f(1, 0, 0);

    //c. specify to draw a point
    glBegin(GL_POINTS);
        glVertex2i (x,y);
    glEnd();

    //d. start drawing
    glFlush();
}
static void reshape(int w, int h) {
    //e. specify the window’s coordinates
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho(0, Width, 0, Height, -1.0, 1.0);
}

int main(int argc, char **argv) {
    //f. initialize a drawing area
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE);
    glutInitWindowSize(Width, Height);
    glutCreateWindow("Example 1.1.point.c");

    //g. specify event callback functions
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutIdleFunc(display);
    glutMainLoop();
}
Building an Executable JAR File

• To facilitate sharing and homework check
• Steps:
  – Set up a working directory to build your jar file.
  – Move all the necessary java and class files into this directory, including all inheritance class files.
  – Create a text file “manifest-info.txt” in the same directory:
    Class-Path: gluegen-rt.jar jogl.jar
    Main-Class: J1_1_Point
  – Execute the following in the same directory from the command line:
    > "C:\myJDK\bin\jar" -cfm myexe.jar manifest-info.txt *.clas
Run the executable jar file

- You need to put the library jar files (jogl.jar and gluegen-rt.jar) in the same directory. You may want to put all the dll files in the same directory as well. Your directory will contain the following files as in our example:

  2008-01-07 08:31   <DIR>        ..
  2008-01-07 09:19   1,766 J1_1_Point.class
  2008-01-06 18:10   2,190 J1_0_Point.class
  2008-01-07 09:19   736 J1_1_Point$1.class
  2008-01-07 09:46   61 manifest-info.txt
  2008-01-07 09:46   3,419 myexe.jar
  2007-04-22 02:00   1,065,888 jogl.jar
  2007-04-22 02:00   17,829 gluegen-rt.jar
  2007-04-22 02:00   20,480 gluegen-rt.dll
  2007-04-22 02:00   315,392 jogl.dll
  2007-04-22 02:00   20,480 jogl_awt.dll
  10 File(s)        1,448,241 bytes

- Now you can either double-click on the jar file or execute it on a command line with:

  > "C:\myJDK\bin\java" -jar myexe.jar
APPLICATIONS

• **Graphical User Interface (GUI):** desktop window systems (menu items, icons, objects)

• **Science & Engineering:** visualization, modeling, simulation

• **Medicine:** medical imaging, surgery planning

• **Art, Commerce, Entertainment:** galleries, games, ads, film (Jurassic Park, Star Trek)

• **Computer-aided design (CAD):** buildings, auto bodies, airplane wings, VLSI chips, etc.

• **Office automation and electronic publishing**

• **Cartography and information visualization**
BRIEF HISTORY

• 1963 Evan Sutherland’s seminal doctoral work “Sketchpad: a man-machine graphical communication system”
• 1977 3D Core Graphics System by ACM SIGGRAPH Committee
• 1980’s, hardware expensive & graphics app. programs were few
• 1985 GKS (the Graphical Kernel System), 2D
• 1987 MIT’s X lib for windows under UNIX OS
• 1988 GKS-3D, PHIGS (ANSI and ISO standard); SGI’s GL;
• 1993 OpenGL (Open standard from SGI)
• Since then, OpenGL supported on PC, MAC, Sun work-stations, SGI work-stations, special purpose image generators (ESIG)
• Since then, Evolution of advanced GUI’s and graphics environments; Software from low-level, device dependent packages to higher-level device-independent packages

• 2003 JOGL – Java for OpenGL from Sunmicrosystem
The graphics system generates the image specified by the application program in the framebuffer and displays the picture on the display.

The application program accepts user input and controls the graphics model (create, store, retrieve, modify, and send it to the graphics system).

The digitization process is called scan-conversion (drawing, rendering).
• Mouse, keyboard for **input**, CRT (cathode ray tube) for **output** (Other input/output devices)

• Most of the programmer’s task concerns creating and changing the **model** and handling **user interaction**

• Models typically store **descriptions** of the shapes, positions, attributes (color), and hierarchical relationships of the components

• A **graphics library** includes **output subroutines of primitives** and attributes, such as `glRecti(x1,y1,x2,y2)` , `glColor3f(r,g,b)`, etc.
  ```c
  glBegin(GL_POINTS);
  glVertex2i (x,y);
  glEnd();
  ```
• The **screen** is a window for the user to manipulate not only the image, but also the model

• A picture is produced as an array (raster) of picture elements (pixels) saved in the **framebuffer**

• The **framebuffer** corresponds to the CRT picture.

• An **event** is a user input or other state change queued with other events generated in timely order

• Each event is handled in the queued order by the system or user **event handler** (user program).
Complex Object and Animation

• To draw a complex shape, we need an application program to dissect it into pieces of simple shapes (primitives), or construct it from primitives.

• We have the magic pen that draws a pixel. If we can draw a pixel, we can draw a point, a line, a triangle (polygon), a curve, a surface, …

• We can draw a complex object, clear the drawing area, draw the object at a slightly different location, and repeat the above processes — the object is animated.
HW0: due before next class

1. Set up your programming environment, and run the sample programs.
2. If you failed, meet the TA or come to my office during my office hours or bring your computer to the next class.

• I won’t be happy to receive questions about the working environment after this week.