• The OpenGL graphics system is an application Programming interface (API) to graphics hardware.

• Around 130 graphics drawing and operation functions
**OpenGL-Related Libraries**

- The OpenGL Utility Library (GLU) contains several routines that use OpenGL commands. It is considered part of your OpenGL.
- The OpenGL Extension to the X Window System (GLX) provides a means of creating an OpenGL context and associating it with a drawable window on a machine that uses the X.
- **WGL** for Windows; **AGL** for Apple; **PGL** for OS/2
- **OpenGL Auxiliary Library (AUX) or Utility Toolkit (GLUT)**
  - to make programming examples simpler. Works for X, Win, Apple, etc.
- **JOGL** – Java OpenGL API
/* Example 1.1.point.c: draw randomly generated points */

#include <stdlib.h>
#include <GL/glut.h>

#define Height 400
#define Width 400
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();
}
/* Example 1.1.point.c: revisit in JOGL */

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

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
        //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
}
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();

    // 6. Add a title on the frame
    f.setTitle("JOGL J1_1_Point");
    f.setSize(WIDTH, HEIGHT);
    f.setVisible(true);
}
}
OpenGL and DirectX libraries

• Theoretically, OpenGL and DirectX (Direct3D) are the same

• OpenGL is a mature graphics library, while DirectX has been advanced rapidly and significantly

• OpenGL is easier to learn and use, and is integrated in most modern graphics textbooks

• DirectX is much more popular in PC game Industry
**JOGL – Java for OpenGL**

- **JOGL** implements Java bindings for OpenGL.
- Provides platform for 3D graphics applications in Java
- JOGL provides full access to OpenGL functions and integrates with the AWT and Swing widget sets.
- Part of a suite of open-source technologies initiated by the Game Technology Group at Sun Microsystems.
ANIMATION

• Motion in a movie is achieved by projecting a sequence of pictures at 24 frames per second on the screen

• 60 per second is smoother than 30, and 120 is marginally better than 60. Refresh rates faster than 120 make no difference

    open_window();
    for (i = 0; i < 1000000; i++) {
        clear_the_framebuffer ();
        draw_frame(i);
        wait_until_a_24th_of_a_second_is_over ();
    }
• Items drawn first are visible for the full 1/24 second and present a solid image on the screen; items drawn toward the end are instantly cleared as the program starts on the next frame.

• **Double-buffering** - hardware or software that supplies two complete color buffers. The *frontbuffer* is displayed while the *backbuffer* is drawn. When the drawing of a frame is complete, the two buffers are swapped.

```c
open_window_in_double_buffer_mode();

for (i = 0; i < 1000000; i++) {
    clear_the_backbuffer ();
    draw_a_frame_into_backbuffer (i);
    swap_the_buffers (); // which is mostly implied
}
```
glFlush()

• You treat a double-buffer as a single-buffer by setting `glDrawBuffer()` to `GL_FRONT` with a `glFlush()` call.

• To switch back to double-buffered, you need to set `glDrawBuffer()` to `GL_BACK`.

• You can just choose to draw into both buffers:
  `glDrawBuffer(GL_FRONT_AND_BACK);`
Some Examples Coded In OpenGL
Vector Operations

- Vector: a and b
  - a=(a0, a1, a2); b=(b0, b1, b2);
  - It gives a length and a direction
    - Length: |a| = sqrt(a0*a0 + a1*a1 + a2*a2)
    - Direction: a/|a| = (a0 /|a| , a1 /|a| , a2 /|a| )
      // normalize a vector to unit vector

```java
public void normalize(double a[]) {
    double d = Math.sqrt(a[0]*a[0] + a[1]*a[1] + a[2]*a[2]);
    if (d == 0) {System.err.println("0 length vector: normalize()."); return;}
}
```

Copyright @ 2012 by Jim X. Chen:
jchen@gmu.edu
Vector Operations – Cont.

- **Addition/Subtraction**
  - \( a+b = (a_0+b_0, a_1+b_1, a_2+b_2) \);

- **Dot Product**
  - \( a \cdot b = |a||b|\cos\theta = a_0b_0 + a_1b_1 + a_2b_2 \)
  - A scalar value

```java
public double dotprod(double[] a, double[] b) {
}
```
Vector Operations – Cont.

• Cross Product
  – $\mathbf{a} \times \mathbf{b} = \mathbf{a}$ vector perpendicular to $\mathbf{a} \& \mathbf{b}$
  – $\mathbf{a} \times \mathbf{b} = \mathbf{n}|\mathbf{a}||\mathbf{b}||\sin\theta$ // cross product of two vectors

```java
public void crossprod(double[] a, double[] b, double[] v) {
}
```

Copyright © 2012 by Jim X. Chen:
jchen@gmu.edu
Reflect $v_1$ around $n$ to $v_2$

- $n$ is normalized;
- $\text{dotprod}(v_1,n)$ is the projection length of $v_1$ on $n$
- $2 \times \text{dotprod}(v_1,n)$ is twice of the above
- $2 \times \text{dotprod}(v_1,n) \times n$ is the vector along $n$ with the length
- The above minus $v_1$ will give you $v_2$
// reflect v1 around n to v2
public void reflect(double v1[], double n[], double v2[]) {

    // v2 = 2*dot(v1, n)*n + v1
    for (int i = 0; i < 3; i++) {
        v2[i] = 2 * dotprod(v1, n) * n[i] - v1[i];
    }
}
HW1: due before next class

- draw points that moves slowly along a circle
- Draw multiple points that move and bounce inside the circle
- Draw multiple points that bounces outside a circle and inside a rectangle