A Graphics System

- **Application program**
  - Puts all subsequent layers together into a piece of software that is of direct use to some human being
- **Framework**
  - Successive layers over a core graphics engine to make it easier to use
  - User interfaces, image processing libraries, 3D scene specifiers live here
- **Graphics engine**
  - Software specific to graphics rendering: pipelines, special algorithms (such as full-screen anti-aliasing)
- **Operating system**
  - Manages the context of graphics applications: windows, full-screen mode, drag-and-drop
  - If properly done, is a cleanly separate layer from core operating system functions: processes, memory management, file systems, I/O
- **Device driver**
  - Software specific to the video hardware
  - Interfaces with the operating system in a standard way
  - May also have hooks to a graphics framework (i.e. OpenGL)
- **Video hardware**
  - Video memory, registers
  - Coprocessors, special functions
  - Interface to output devices
- **Input devices**
  - Allows the user to affect software: physical activity such as movement travels through the layers of a system until something figures out what to do with it

An OpenGL Graphics System

- **Application program**
  - Written in C, C++, Java, or any other programming language for which OpenGL can be “exposed”
- **Framework**
  - GLUT: OpenGL Utility Toolkit — platform-independent windowing framework specifically for OpenGL software, available on multiple platforms...results in portable OpenGL software
  - Platform-specific frameworks (xgl, wgl, agl) — binds OpenGL to APIs that are specific to an operating system or platform...not portable, but may take special advantage of particular platform strengths
- **Graphics engine**
  - OpenGL!!!
- **Operating system**
  - Put your favorite operating system here
  - Special mention: in Mac OS X, OpenGL is fundamental to the operating system — its graphical user interface is implemented in OpenGL; in other operating systems, OpenGL rides “on top of” what the operating system provides
- **Device driver**
  - Put your favorite nVidia, ATI, or other driver here
- **Video hardware**
  - Ditto, but for graphics cards
- **Input devices**
  - Keyboards, mice, trackballs, gloves, tablets, oh my!
A Java/2D/3D Graphics System

- Application program
  - Written in Java
- Framework
  - AWT/Swing: Java’s windowing and user interface toolkit
  - Java2D: Java’s base 2D graphics library; built-in
  - Java Advanced Imaging: augments Java2D with image processing routines; add-on
  - Java3D: Scene-based 3D API; add-on
- Graphics engine
  - Frequently OpenGL under the hood, but not necessarily
- Operating system
  - Put your favorite Java-capable operating system here
- Device driver
  - Put your favorite nVidia, ATI, or other driver here
- Video hardware
  - Ditto, but for graphics cards
- Input devices
  - Keyboards, mice, trackballs, gloves, tablets, oh my!

Getting into OpenGL

- OpenGL is focused on 3D drawing; it is not concerned with user interface-level constructs like windows, widgets, etc. Leave it to the operating system or framework to give OpenGL a “canvas” on which it can do its 3D magic
- OpenGL lives in an abstract 3D coordinate system which is right-handed by default
- Two layers:
  - GL — core OpenGL, this is the basic library
  - GLU — OpenGL Utility Library: very useful routines, but all a layer above GL; what Java calls “convenience methods”
- For the third layer — the context — there are 2 ways to go
  - Portable = GLUT — OpenGL Utility Toolkit
  - Platform-specific = xGL — Platform-specific OpenGL interfaces (xgl for X-windows, wgl for Windows, agl for Mac OS)
- OpenGL is a state machine
Coding Details

- 3 sets of header files and 3 libraries, corresponding to GL, GLU, and GLUT; content is the same, but filenames/locations vary by platform
- Once properly set-up, your code can say:
  - `#include <GL/gl.h>`
  - `#include <GL/glu.h>`
- But, if you’re using GLUT, it’s simpler; GLUT includes gl.h and glu.h for you, so you only have to say:
  - `#include <GL/glut.h>` - or -
  - `#include <GLUT/glut.h>` (depending on your implementation)
- If compiling off the command line or a make file, you need to mention the include and library directories, for example:
  - `gcc -I/usr/X11R6/include -L/usr/X11R6/bin -o myproject myproject.c -lglut -lGLU -lGL`
- In an IDE, this should not be necessary as long as you “tell” your IDE that you need OpenGL

Sample OpenGL Setups

<table>
<thead>
<tr>
<th>GNU compiler under Linux (Red Hat 6.2)</th>
<th>GNU compiler under Mac OS X</th>
<th>GNU compiler under Windows</th>
<th>Microsoft compiler under Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/usr/X11R6/include/GL</code></td>
<td><code>/System/Library/Frameworks/OpenGL.framework/Versions/Current/Headers</code></td>
<td><code>/gcc/include/gl</code></td>
<td><code>%WINSYS%</code></td>
</tr>
<tr>
<td><code>/usr/X11R6/lib</code></td>
<td><code>/System/Library/Frameworks/OpenGL.framework/Versions/Current/Libraries</code></td>
<td><code>/gcc/lib/gcc-lib</code></td>
<td><code>%VC%</code></td>
</tr>
<tr>
<td><code>gl.h</code></td>
<td><code>libGL.dylib</code></td>
<td><code>libgl.h</code></td>
<td><code>%WINSYS%</code></td>
</tr>
<tr>
<td><code>libGLU.a</code></td>
<td><code>libGLU.dylib</code></td>
<td><code>libGLU.dylib</code></td>
<td></td>
</tr>
<tr>
<td><code>glut.h</code></td>
<td><code>libGLU.dylib</code></td>
<td><code>libglut32.a</code></td>
<td><code>opengl32.dll</code></td>
</tr>
</tbody>
</table>

Note that these differences don’t affect your source code! It’s still:

```
#include <GL/gl.h>
...etc.
```
The main() function

```c
int main(int argc, char **argv) {
    glutInit(&argc, &argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(250, 250);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Icosahedron");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutMotionFunc(motion);
    glutKeyboardFunc(keyboard);
    glutSpecialFunc(special);
    init();
    glutMainLoop();
}
```

The main() function

The event handlers: view

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(0.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glRotatef(spin, 0.0, 0.0, 1.0);
    glBegin(GL_TRIANGLES);
    for (int t = 0; t < 20; t++) {
        glColor3f(1.0, 1.0, 1.0);
        gNormall3f(vdata[tindices[1][0]][0]);
        gNormall3f(vdata[tindices[1][0]][1]);
        gNormall3f(vdata[tindices[1][0]][2]);
        glVertex3f(vdata[tindices[1][1]][0]);
        glVertex3f(vdata[tindices[1][1]][1]);
        glVertex3f(vdata[tindices[1][1]][2]);
    }
    glEnd();
    glutSwapBuffers();
    glFlush();
}
```

The function assigned to glutDisplayFunc() draws the scene. In general, this means (1) clearing the scene, (2) setting up the camera, (3) drawing the objects in the scene, then (4) cleaning up.

```
void reshape(int w, int h) {
    glViewport(0, 0, 250, 250);
    double aspect = (double)w / (double)h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (h > w) {
        glFrustum(-1.0, 1.0, -aspect, aspect, 1.5, 20.0);
    } else {
        glFrustum(-1.0 / aspect, 1.0 / aspect, -1.0, 1.0, 1.5, 20.0);
    }
}
```

The function assigned to glutReshapeFunc() is responsible for tweaking anything that depends on the current size/shape/proportions of the window.
Anatomy of an OpenGL GLUT program, a.k.a. “GLUT guts”  
(see Chapters 1-2, OpenGL Programming Guide)

The event handlers: controller

```c
void spinDisplay(void) {
    spin = spin + 0.01;
    if (spin > 360.0)
        spin = spin - 360.0;
    glutPostRedisplay();
}

void mouse(int button, int state, int x, int y) {
    switch(button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN) {
                if (spinOn)
                    glutIdleFunc(NULL);
                else
                    glutIdleFunc(spinDisplay);
                spinOn = !spinOn;
            } break;
        default: break;
    }
}
```

The function assigned to glutIdleFunc() is called whenever the GLUT event loop “has some time.” Typically it tweaks the model and invokes a redisplay of the scene.

The function assigned to glutMouseFunc() responds to button clicks, and should have this signature. Typically, it interprets the user’s mouse activity and tweaks the model accordingly. It may or may not request a redisplay of the scene.

All other functions that respond to user activity — glutMotionFunc(), glutKeyboardFunc(), glutSpecialFunc(), among others — work in a similar manner.

Anatomy of an OpenGL GLUT program, a.k.a. “GLUT guts”  
(see Chapters 1-2, OpenGL Programming Guide)

Your code: the model and initialization

```c
static GLfloat spin = 0.0;
static int spinOn = 0;
static const GLfloat X = 0.525731112119133606;
static const GLfloat Y = 0.850650808352039932;
static const GLfloat Z = 0.525731112119133606;

static const GLfloat tindices[20][3] = {
    { X, 0.0, Z }, { X, 0.0, -Z }, { -X, 0.0, Z }, { -X, 0.0, -Z },
    { 0.0, Z, X }, { 0.0, -Z, X }, { Z, 0.0, X }, { -Z, 0.0, X },
    { 0.0, 0.0, Z }, { 0.0, 0.0, -Z }, { X, 0.0, 0.0 }, { -X, 0.0, 0.0 },
    { Z, X, 0.0 }, { Z, -X, 0.0 }, { -Z, -X, 0.0 }
};

static const GLuint tindices[20][3] = {
    { 1, 4, 0 }, { 4, 9, 0 }, { 4, 5, 9 },
    { 5, 1, 4 }, { 1, 6, 4 }, { 1, 10, 4 },
    { 10, 3, 8 }, { 8, 2, 3 }, { 3, 2, 5 },
    { 5, 7, 2 }, { 3, 10, 7 }, { 10, 6, 7 },
    { 6, 11, 7 }, { 4, 11, 1 }, { 4, 1, 10 },
    { 10, 1, 6 }, { 11, 0, 9 }, { 2, 11, 9 },
    { 5, 2, 9 }, { 11, 2, 7 }
};

void init(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);
}
```

This is where you must need good design. This sample code can only get away with a bunch of statics because it is very simple. Anything past this, and you should start defining classes and putting them in separate files.

You don’t really need a separate initialization function, but it’s good practice anyway. It is a clear place for one-time or initial setup code, whether for the model or for OpenGL settings. For example, instead of hard-declaring the icosahedron’s vertices and faces above, those arrays could have been calculated (or read from a file, or whatever) in here.