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 APIs 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 knows how to respond to 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 the actual 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.

- You can simulate 2D — just lock the z coordinate at 0.0 (that’s what OpenGL’s “2D” functions do).

Two layers:

- GL — 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, etc.)

- **OpenGL is a state machine**
Making Your Code and Compiler “See” OpenGL

• GL, GLU, and GLUT → 3 header files and 3 libraries

• Thus, one might expect these:
  
  ```
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
  ```
  — or —
  ```
  #include <GLUT/glut.h>
  ```
  (depends on platform)

• But conveniently, GLUT includes gl.h and glu.h for you, so you only need the last `#include` directive

• If compiling off the command line or a make file, you may or may not need to mention the include and library directories:

  ```
  gcc -I/usr/X11R6/include -L/usr/X11R6/bin -o myproject myproject.c -lGLU -lGL -lglut
  ```

  ```
  gcc -o myproject myproject.c -lopengl32 -lglu32 -lglut32
  ```

• On Mac OS X, OpenGL and GLUT are packaged as “frameworks,” which are referenced differently in the compiler command:

  ```
  gcc -o myproject myproject.c -framework OpenGL -framework GLUT
  ```

• In an IDE, you need to “tell” your IDE where the OpenGL include files and libraries are

  ```
  gcc -o myproject myproject.c -framework Foundation -framework OpenGL -framework GLUT
  ```

  Older versions of these frameworks also depended on the Foundation framework, so you may need to add that too:

  ```
  gcc -o myproject myproject.c -framework Foundation -framework OpenGL -framework GLUT
  ```
Sample OpenGL Setups

<table>
<thead>
<tr>
<th>GNU compiler under Linux (Ubuntu/Debian)</th>
<th>/usr/include/GL</th>
<th>/usr/lib</th>
<th>/usr/lib</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL</td>
<td>gl.h</td>
<td>libGL.la</td>
<td>libGL.so</td>
</tr>
<tr>
<td>GLU</td>
<td>glu.h</td>
<td>libGLU.la</td>
<td>libGLU.so</td>
</tr>
<tr>
<td>GLUT</td>
<td>glut.h</td>
<td>libglut.a</td>
<td>libglut.so</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Xcode (GNU) compiler under Mac OS X</th>
<th>/System/Library/Frameworks/OpenGL.framework/Versions/Current/Headers</th>
<th>/System/Library/Frameworks/OpenGL.framework/Versions/Current/Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL</td>
<td>gl.h</td>
<td>libGL.dylib</td>
</tr>
<tr>
<td>GLU</td>
<td>glu.h</td>
<td>libGLU.dylib</td>
</tr>
<tr>
<td>GLUT</td>
<td>/System/Library/Frameworks/GLUT.framework/Versions/Current-Headers/glut.h</td>
<td>No explicit library — implemented through the operating system.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GNU compiler under Windows (e.g., MinGW)</th>
<th>C:\mingw\include\gl</th>
<th>C:\mingw\lib</th>
<th>%WINSYS%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL</td>
<td>gl.h</td>
<td>libopengl32.a</td>
<td>opengl32.dll</td>
</tr>
<tr>
<td>GLU</td>
<td>glu.h</td>
<td>libglu32.a</td>
<td>glu32.dll</td>
</tr>
<tr>
<td>GLUT</td>
<td>glut.h</td>
<td>libglut32win.a</td>
<td>glu32.dll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Microsoft compiler under Windows</th>
<th>%VC%/include</th>
<th>%VC%/lib</th>
<th>%WINSYS%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL</td>
<td>gl.h</td>
<td>opengl32.lib</td>
<td>opengl32.dll</td>
</tr>
<tr>
<td>GLU</td>
<td>glu.h</td>
<td>glu32.lib</td>
<td>glu32.dll</td>
</tr>
<tr>
<td>GLUT</td>
<td>glut.h</td>
<td>glu32.lib</td>
<td>glu32.dll</td>
</tr>
</tbody>
</table>

Note that these differences don’t affect your source code — it’s still:

```c
#ifdef __APPLE_CC__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
```
Anatomy of an OpenGL GLUT program, a.k.a. “GLUT guts” (see Chapters 1-2, *OpenGL Programming Guide*)

**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(mouse);
    glutKeyboardFunc(KeyboardEvent);
    glutSpecialFunc(keypress);
    init();
    glutMainLoop();
}
```

**Event handlers:** pass a function. Only `glutDisplayFunc` is required; all others are optional, including:

- `glutIdleFunc`, `glutKeyboardFunc`, `glutMotionFunc`, `glutMenuStateFunc`, `glutSpecialFunc`
- and many more.

**Window setup (for single-window programs)**

- Configures the display; choose from:
  - GLUT_SINGLE, GLUT_DOUBLE
  - GLUT_RGB, GLUT_RGBA, GLUT_INDEX
  - GLUT_ACCUM, GLUT_ALPHA, GLUT_DEPTH, GLUT_STENCIL, GLUT_MULTISAMPLE, GLUT_STEREO, GLUT_LUMINANCE

**Event handlers:** pass a function. Only `glutDisplayFunc` is required; all others are optional, including:

- `glutIdleFunc`, `glutKeyboardFunc`, `glutMotionFunc`, `glutMenuStateFunc`, `glutSpecialFunc`
- and many more.

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.

**The event handlers: view — displaying**

```c
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_TRIANGLES);
    for (int i = 0; i < 20; i++) {
        glVertex3fv(&vdata[tindices[i][0]][0]);
        glVertex3fv(&vdata[tindices[i][0]][0]);
        glVertex3fv(&vdata[tindices[i][0]][0]);
    }
    glEnd();
    glutSwapBuffers();
    glFlush();
}
```
Anatomy of an OpenGL GLUT program, a.k.a. “GLUT guts”
(see Chapters 1-2, OpenGL Programming Guide)

The event handlers: view — resizing

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

Food for thought: What is the significance of the window’s aspect ratio — the proportion of its width vs. height — with respect to the scene being displayed?

The function assigned to glutReshapeFunc is responsible for tweaking anything that depends on the current size/shape/proportions of the window, which are passed as integers w and h.

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)

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;
    }
}
```
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 Z = 0.850650808352039932;
static GLfloat vdata[12][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 },
    { 0.0, -Z, X }, { 0.0, -Z, -X }, { Z, X, 0.0 },
    { -Z, X, 0.0 }, { Z, -X, 0.0 }, { -Z, -X, 0.0 }
};

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

Your code: initialization

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