CMSC 427: Chapter 2
Graphics Libraries and OpenGL

Reading: (Not really covered in our text. See Sects 18.1, 18.2.)
Overview:
- Graphics Libraries
- OpenGL and its Structure
- Drawing Primitives in OpenGL
- GLUT and Event-Driven Programming
- Buffers and raster operations

The Graphics Pipeline

Geometry Processing
- Model (description)
- Retrieve (database)
- Transform (eye space)
- Clip (viewing volume)
- Light (for each spectral component)

Image Processing
- Scan (primitives to pixels)
- Visibility (at each pixel)
- Texture (texture map)
- Composite (masks, other images)
- Frame Buffer

Display (CRT/LCD/DMD)
Types Graphics Libraries

Retained-Mode Libraries (e.g., Java3d, Ogre)
- Data is copied to library’s internal data-structures.
- Library’s copy is sent down each frame.
- Library can perform global optimization.
- Less suitable for time-varying datasets.
- Functionally analogous to program compilation.

Immediate-Mode Libraries (e.g., OpenGL, DirectX)
- Application supplies all the primitives for display every frame.
- Library can optimize only per-primitive (local optimizations).
- Only the application can perform global optimizations.
- More suitable for scenes where changes happen every frame.
- Functionally analogous to program interpretation.
Overview

- OpenGL Architecture
- OpenGL Objects and Attributes
- Event-Driven Programming
- GLUT
- OpenGL/GLUT Sample Program
- OpenGL Buffers
- OpenGL Pixel Operations

OpenGL: http://www.opengl.org

Standard: Most widely-used/supported 2D/3D graphics API
- Windows NT/95/98/00, UNIX, Linux, MacOS, OS/2, Python, ...
- Bindings for C, C++, Java, Fortran, Ada
- ATI, HP/Compaq, E&S, IBM, Intel, Intergraph, NVIDIA, Microsoft, SGI

Independent of hardware, OS, window system.

Immediate-mode (mostly): Design philosophy is immediate-mode, with a few retained-mode structures. Appearance attributes have per-frame state.

Windowing not included: Does not include commands for windowing tasks or user interaction. (This is what GLUT is for.)
API Hierarchy

OpenGL works with different operating systems and windowing systems by interfacing with the operating system and window-specific display libraries:

GLU (OpenGL Utilities): provides transformation matrix utilities, common objects (sphere, cylinder), implicit primitives (quadrics), parametric primitives (NURBS), utilities for image manipulation

OpenGl Architecture

Other Features:
- Supports imaging as well as 2D/3D geometry operations
- Accepts information on transformation, lighting, texturing, blending, anti-aliasing, etc.
- Can return information about all state variables (including queries to frame buffer and texture memory)
Overview

• OpenGL Architecture
• OpenGL Objects and Attributes
• Event-Driven Programming
• GLUT
• OpenGL/GLUT Sample Program
• OpenGL Buffers
• OpenGL Pixel Operations

OpenGL Naming Conventions

Functions: begin with gl
Constants: begin with GL_
Types: begin with GL

<table>
<thead>
<tr>
<th>Suffix</th>
<th>OpenGL Datatype</th>
<th>C/C++ Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>GLbyte</td>
<td>signed char</td>
</tr>
<tr>
<td>s</td>
<td>GLshort</td>
<td>short</td>
</tr>
<tr>
<td>i</td>
<td>GLint</td>
<td>int</td>
</tr>
<tr>
<td>ub</td>
<td>GLubyte</td>
<td>unsigned char</td>
</tr>
<tr>
<td>us</td>
<td>GLushort</td>
<td>unsigned short</td>
</tr>
<tr>
<td>ui</td>
<td>GLuint</td>
<td>unsigned int</td>
</tr>
<tr>
<td>f</td>
<td>GLfloat</td>
<td>float</td>
</tr>
<tr>
<td>d</td>
<td>GLdouble</td>
<td>double</td>
</tr>
</tbody>
</table>
Specifying Vertices

**Vertex Arguments:** All objects in OpenGL are constructed from convex polygons, which are represented by their vertex coordinates. The argument type is specified by the suffix to the OpenGL function name:

```latex
(func\_name) \langle \text{dim} \rangle \langle \text{type} \rangle (\langle \text{argument list} \rangle)
```

**Examples:**
- 2D point in GLint (int) coordinates:
  ```latex
  glVertex2i (200, -150);
  ```
- 3D point in GLfloat (float) coordinates:
  ```latex
  glVertex3f (200.3f, -150f, 40.75f);
  ```

**Vector (array) Arguments:** Add suffix "v" to the function name

- 3D point in GLdouble (double) coordinates given as a vector:
  ```latex
  GLdouble pt[3] = { 200.3, -150, 40.75 };
  glVertex3dv (pt);
  ```

OpenGL Objects

OpenGL supports a number of different complex objects. Each object is specified by enumerating the vertices that define the object.

**General Form:**

```latex
glBegin (\langle \text{object type} \rangle);
  glVertex \ldots (\ldots);
  glVertex \ldots (\ldots);
  glVertex \ldots (\ldots);
 glEnd ();  \quad // \text{don't forget this!}
```

**Note:** There are a number of other attributes that can be placed within the `glBegin`...`glEnd` pair. These affect things like **color** and shading properties. We will discuss these later.
Isolated Points

**GL_POINTS**: Draws a set of isolated points.

```c
glBegin ( GL_POINTS );
    glVertex2i ( 0, 0 );
    glVertex2i ( 0, 1 );
    glVertex2i ( 1, 0 );
    glVertex2i ( 1, 1 );
glEnd ( );
```

---

Line Loop (Polyline)

**GL_LINE_LOOP**: Draws a closed polygonal line (segments joined end to end).

```c
glBegin ( GL_LINE_LOOP );
    glVertex2i ( 0, 0 );
    glVertex2i ( 0, 1 );
    glVertex2i ( 1, 1 );
    glVertex2i ( 1, 0 );
glEnd ( );
```

**Variants:**

- **GL_LINE_STRIP**: Polygonal line, but not closed off to form a loop.
- **GL_LINES**: A sequence of line segments, not connected to each other.
**Polygon**

**GL_POLYGON**: Draws a filled convex polygon.

```c
glBegin ( GL_POLYGON );
    glVertex2i( 0, 0 );
    glVertex2i( 0, 1 );
    glVertex2i( 1, 1 );
    glVertex2i( 1, 0 );
glEnd( );
```

*Note: OpenGL assumes that all polygons are **convex**, meaning that all interior angles are at most 180 degrees. OpenGL is largely "silent" about errors, so be careful.*

---

**Triangles**

**GL_TRIANGLES**: Draws a series of filled triangles. Each sequence of three vertices defines a separate triangle.

```c
glBegin ( GL_TRIANGLES );
    glVertex2i( 0, 1 );
    glVertex2i( 0, 0 );
    glVertex2i( 1, 1 ); // abc
    glVertex2i( 1, 1 );
    glVertex2i( 0, 0 );
    glVertex2i( 1, 0 ); // cbd
    glVertex2i( 1, 0 );
    glVertex2i( 2, 0 );
    glVertex2i( 1, 1 ); // dec
    glVertex2i( 2, 0 );
    glVertex2i( 1, 1 );
    glVertex2i( 2, 0 );
glEnd ( );
```

*Note: It is a good idea enumerate all vertices in a consistent order, say, *counterclockwise*. We will discuss why later.*
Triangle Strip

**GL_TRIANGLE_STRIP:** Draws a series of triangles by joining the next vertex to the previous two.

```c
glBegin ( GL_TRIANGLE_STRIP );
    glVertex2i ( 0, 1 ); // a
    glVertex2i ( 0, 0 ); // b
    glVertex2i ( 1, 1 ); // c→abc
    glVertex2i ( 1, 0 ); // d→bcd
    glVertex2i ( 2, 0 ); // e→cde
glEnd ( );
```

**Orientation:** Note that orientations appear to alternate (abc is CCW, bcd is CW, and cde is CCW). OpenGL automatically reverses the orientation of every other triangle, so they all will be consistent with the first. In this case, all will be CCW.

---

In-Class Exercise

Draw the shape on the right using a **single triangle strip**, with triangles drawn in counter-clockwise orientation.

Drawn the shape on the right using a **small number of triangle strips**.

(What is the smallest number you can find?)
Attributes

**Attributes**: Affect the manner in which objects are drawn.
- These can be placed within each `glBegin…glEnd` pair.
- Once set, they affect subsequent objects, until changed again.

**Point**:
- Point size: `glPointSize(2.0);`
- Point color: `glColor3f(0.0, 0.0, 1.0);` (Sets RGB color components: Red, Green, Blue.)

**Line**:
- Line width: `glLineWidth(2.0);`
- Line color: `glColor3f(0.0, 0.0, 1.0);`

**Face**:
- Front and/or back: `GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`
- Face color: `glColor3f(0.0, 0.0, 1.0);`

Overview

- OpenGL Architecture
- OpenGL Objects and Attributes
  - Event-Driven Programming
- GLUT
- OpenGL/GLUT Sample Program
- OpenGL Buffers
- OpenGL Pixel Operations
Event-Driven Computing

Typical (noninteractive) Program:
- Read Data.
- Process Data.
- Output Results.

Event-Driven Computing: (System's perspective)
- Check whether an event has occurred.
- if (an event has occurred)
  • call event_handler(event_type).
- Repeat above.

Event-Driven Computing: (Programmer's perspective)
- Register "event-handler" pairs. (For each "event" call a function that performs "handles" this event and returns.)
- Pass control to the operating system.

Event Queue

Operating System/Window Management System: Copies all handled events to an Event Queue.

Let EQ be the event queue (first-in, first-out queue).

New event(e):
if (e is handled) { append e to the end of EQ }

Processing Events:
while (true) {
  if (EQ is not empty) {
    call event_handler(EQ.front());
    remove front event from EQ;
  }
}
Overview

• OpenGL Architecture
• OpenGL Objects and Attributes
• Event-Driven Programming
  • GLUT
• OpenGL/GLUT Sample Program
• OpenGL Buffers
• OpenGL Pixel Operations

OpenGL and GLUT

**OpenGL is window-system independent**
- Makes it portable.
- Can be targeted for **different platforms**: PCs, game consoles, interactive TV set-top boxes.
- Independent of:
  - operating system
  - windows system
  - display size and display properties (only assumes a raster-device).

**But**: OpenGL still needs to **interface** with windowing system.
  For instance: X-windows, Microsoft’s windows system

**GLUT (to the rescue)**: GLUT (GL Utility Toolkit) is a window-system independent programming **interface** for OpenGL.
**GLUT: OpenGL Utility Toolkit**

**GLUT:**
- is a simple programming interface to the windows system.
- is largely window-system independent.
- supports only pop-up menus (no pull-down menus).
- maintains its own event loop.
- accepts registration of callback functions from user programs.
- has its own (limited) set of fonts.

**Some useful sites for GLUT:**
- Mark Kilgard’s GLUT page: Full online documentation.
- Nate Robins GLUT for Win32: Has precompiled binaries for Microsoft Windows and installation instructions!

**GLUT Initialization**

```c
 glutInit ( int* argc, char** argv )
 - Initialize GLUT library. Must be called first.
 - Recognizes and processes GLUT-specific command-line arguments.
```

```c
 glutInitDisplayMode ( unsigned int mode )
 - Specifies a number of options affecting general operation
   connected by "boolean or": E.g. GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE, etc...
 - Examples of Modes:
   GLUT_RGB: Use RGB color (most common)
   GLUT_RGBA: Use RGB + A color (we’ll discuss this later)
   GLUT_DEPTH: Use a depth buffer for hidden surface removal
   GLUT_DOUBLE: Do double-buffering (recommended for smooth animation).
```
**GLUT Window Manipulation**

- **glutInitWindowSize** (int width, int height)
  - Specifies the desired window size. (No guarantees that you will get this.)

- **glutInitWindowPosition** (int x, int y)
  - Requests location (in pixels) of the upper left corner of the window. Note that (0,0) is the upper left corner of the display.

- **glutCreateWindow** (char* window_name)
  - Request that the window be created. (This initializes the request. The window is not created immediately.)

- **glutReshapeWindow** (int width, int height)
  - Resize the display window using the parameters.

Alternately, you can run in full-screen mode:

- **glutFullScreen** ()
  - Resize window to full screen. (See also **glutEnterGameMode** ()).

- **glutEnterGameMode** ()
  - Same as glutFullScreen, but with further speed enhancements.

---

**GLUT Main Event Loop**

- **glutMainLoop** ()
  - Starts the GLUT event processing loop.
  - Never returns (except through callbacks).
  - Calls registered function callbacks (user-defined event handlers) as appropriate.
  - Should be called at most once.
  - Before calling this, be sure to register your events (or this will be a very uninteresting program). 😊
Sample Program (Part I)

```c
int main ( int argc, char** argv ) {  // program arguments
    glutInit( &argc, argv );  // initialize glut and gl
    // double buffering and RGB
    glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB );
    glutInitWindowSize ( 400, 300 );  // initial window size
    glutInitWindowPosition ( 0, 0 );  // initial window position
    glutCreateWindow ( argv[0] );  // create window

    ... initialize callbacks here (described below) ...

    myInit ( );  // your own initializations
    glutMainLoop ( );  // turn control over to glut
    return 0;  // (make the compiler happy)
}
```

This is never executed!

GLUT Window Management

`glutSwapBuffers ( )`
- Swaps front and back buffers.
- This is part of a process called double-buffering (more on this later).

`glutPostRedisplay ( )`
- Mark the current window as "needing to be redrawn".
- GLUT only redraws the window when you request it.
- Next iteration of the `glutMainLoop` will refresh the window using the display-function callback.
- To achieve a continuous animation, call this function repeatedly (e.g., inside a timer loop or idle loop that wakes up every 1/30 second).
GLUT Callback Registration

**glutDisplayFunc** (void (*func)())
- Call the given function whenever the window needs to be redrawn.
  - For example:
  - When window is **first created**.
  - When the window is revealed because an **overlapping window is removed**.
  - When the program explicitly calls **glutPostRedisplay()**.

**glutReshapeFunc** (void (*)(int width, int height))
- Call the given function whenever the window is resized.
- This is always called when the window is **first created**.
- Many OpenGL commands require that the graphics window exists.
- This is useful as a **signal** that the window (requested in glutCreateWindow()) actually **exists**.

**glutKeyboardFunc** (void (*)(unsigned char key, int x, int y))
- Call the given function when a keyboard key is **hit**.
- The (x, y) arguments indicate where the mouse was when key was pressed.
- See also **glutSpecialFunc()** for shift/control keys.

**glutMouseFunc** (void (*)(int button, int state, int x, int y))
- Call the given function when a mouse button is hit or released.
- Button argument: **GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON**.
- State: **GLUT_UP, GLUT_DOWN**.
- Position (x, y): Relative to upper left corner.
- (We’ll see an example below)
GLUT Callback Registration (cont)

`glutMotionFunc` (void (*func) (int x, int y))
- Mouse motion while button pressed. (x, y) is current mouse position.

`glutPassiveMotionFunc` (void (*func) (int width, int height))
- Mouse motion without button press. (Warning: This can generate lots of events!)

`glutIdleFunc` (void (*func) ( ))
- Called whenever no other events are on the event queue.
- Passing NULL disables this.

`glutTimerFunc` (unsigned int msecs, void (*func) (int value), value)
- Callback every msecs milliseconds (or more): Best effort.
- The value parameter is used for setting multiple alarms.
- Function func called with the specified value parameter.

Sample Program (Part II)

```c
int main ( int argc, char** argv ) { // program arguments
    … (given in Part I) …
    glutDisplayFunc ( myDraw ); // set up callbacks
    glutReshapeFunc ( myReshape );
    glutMouseFunc ( myMouse );
    glutKeyboardFunc ( myKeyboard );
    glutTimerFunc (20, myTimeOut, 0 ); // (see below)
    … (given in Part I) …
}
```

Chapter 2, Slide 33
Copyright © D. M. Mount and A. Varshney
Sample Program (Part III)

```c
void myDraw() { // called to display window
  // ...insert your drawing code here ...
}
void myReshape(int w, int h) { // called if reshaped
  windowWidth = w; // save new window size
  windowHeight = h;
  // ...may need to update the projection ...
  glutPostRedisplay(); // request window redisplay
}
void myTimeOut(int id) { // called if timer event
  // ...advance the state of animation incrementally...
  glutPostRedisplay(); // request redisplay
  glutTimerFunc(20, myTimeOut, 0); // request next timer event
}
```

Sample Program (Part IV)

```c
void myMouse(int b, int s, int x, int y) { // called if mouse click
  switch (b) {
    case GLUT_LEFT_BUTTON:
      if (s == GLUT_DOWN) ... // left button pressed
      else if (s == GLUT_UP) ... // left button released
        break;
      // ...
  }
}
void myKeyboard(unsigned char c, int x, int y) { // keyboard key hit
  switch (c) {
    case 'q':
      exit(0); break;
      // 'q' means quit
    // ...
    break;
  }
}
```
### Sample Program (Part V)

```c
void myDisplay () { // display function
    glClear ( GL_COLOR_BUFFER_BIT ); // clear the window

    glColor3f ( 1.0, 0.0, 0.0 ); // set color to red
    glBegin ( GL_POLYGON ); // draw a diamond
        glVertex2f ( 0.90, 0.50 );
        glVertex2f ( 0.50, 0.90 );
        glVertex2f ( 0.10, 0.50 );
        glVertex2f ( 0.50, 0.10 );
    glEnd ( );

    glColor3f ( 0.0, 0.0, 1.0 ); // set color to blue
    glRectf ( 0.25, 0.25, 0.75, 0.75 ); // draw a rectangle

    glutSwapBuffers ( ); // swap buffers (make visible)
}
```

---

### DEMO
Overview

- OpenGL Architecture
- OpenGL Objects and Attributes
- Event-Driven Programming
- GLUT
- OpenGL/GLUT Sample Program
  - OpenGL Buffers
- OpenGL Pixel Operations

Pixel Buffers and Operations

Pixel Buffers: OpenGL maintains from one to many pixel buffers. These buffers store different types of information and have different functions. We will discuss:

- Buffer concepts: bitmaps, pixmaps, depth.
- OpenGL Buffers: color-, depth-, accumulation-, and stencil buffers.
- Transfer: reading and writing pixel buffers.
- Imaging Operations: user operations, bitblt, blend.
Bitmaps and Pixmaps

**Pixel**: A picture element.

**Bitmap**: A 2D array of single-bit pixels (0/1 or black/white).

**Pixmap**: Stack of bitmaps.
The number of bits per pixel is called its **depth**.

To represent full RGB color, it is sufficient to have **24-bit depth**, 8 bits each for red, green, and blue.
OpenGL Buffers

**Color Buffer:** Stores image color information.
- **RGB:** Red, green, blue
- **RGBA** or **RGBA:** Alpha-channel used for blending operations, such as transparency.

**Depth Buffer:** Stores distance to object pixel.
- Used for hidden surface removal - the closest pixel survives.
- Also called the **Z-buffer** (z-coordinate stores distance).

**Accumulation Buffer:** Used for composing and blending images.
- Useful for achieving affects such as motion blur.

**Stencil Buffer:**
- Useful for masking operations.

**Color Buffer: Indexed Color**

**Indexed Color:** Rather than storing full 24-bits per pixel, a small number (say 256) of representative colors are stored in a color-map table, and each pixel stores an index to this table.
- **Less storage:** 8-bits per pixel versus 24.
- **Somewhat slower:** Color lookup cost.
- **Limited:** Fewer simultaneous colors (typically 256).
- **Color mixing:** Such as dithering needed to approximate colors.
- **Rarely used:** Since memory is cheap. (Still seen in .gif image files.)
Color Buffer: Direct Color

Direct Colors: Full 24-bit color is stored in each pixel.
- More storage: 24-bits per pixel.
- Faster: No lookups required.
- General: Can represent more colors.
- More flexible: Easier to perform interpolations, etc.
- Common: Used on virtually all modern systems.

Typically stored in RGB format. (Easy for devices, not for humans.)

Can also include $\alpha$-value for transparency and blending.

Other Buffers

Stencil Buffer:
- Restrict drawing using a stencil (mask).
- Much like cardboard stencils used in art.

Accumulation Buffer:
- Can hold RGBA and allow multiple overlapping drawings to be blended to form final image.
- Useful for antialiasing, soft shadows, motion blur, ...

Double buffering: Using two buffers to achieve smooth animation transitions.
- Drawing is done to the back buffer.
- Front buffer is displayed.
- Swapping between buffers (glutSwapBuffers) is instantaneous.
Buffers Selection in GLUT

void glutInitDisplayMode ( unsigned int mode ): where mode is the bitwise OR of GLUT display mode bit masks:

- **GLUT_RGBA**: Select an RGBA (direct) color. (Default)
- **GLUT_RGB**: (Same as GLUT_RGBA)
- **GLUT_INDEX**: Select indexed color. (Rarely used)
- **GLUT_SINGLE**: Use single buffering. (Default)
- **GLUT_DOUBLE**: Use double buffering.
- **GLUT_ACCUM**: Allocate space for accumulation buffer.
- **GLUT_ALPHA**: Allocate space for color blending.
- **GLUT_DEPTH**: Allocate space for depth buffer.
- **GLUT_STENCIL**: Allocate space for stencil buffer.

Example:

```c
  glutInitDisplayMode ( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
```

Overview

- OpenGL Architecture
- OpenGL Objects and Attributes
- Event-Driven Programming
- GLUT
- OpenGL/GLUT Sample Program
- OpenGL Buffers
  - **OpenGL Pixel Operations**
Reading and Writing Buffers

\texttt{glReadPixels}(\texttt{x, y, width, height, format, type, *pixels})
\texttt{glRasterPos2i}(\texttt{x, y})
\texttt{glDrawPixels}(\texttt{width, height, format, type, *pixels})
\texttt{glCopyPixels}(\texttt{x, y, width, height, format, type, buffer})

where:

format:
\texttt{GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA,}
\texttt{GL_COLOR_INDEX, GL_DEPTH_COMPONENT, ...}

type:
\texttt{GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, GL_FLOAT, ...}
\texttt{GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_SHORT_5_6_5, ...}

buffer:
\texttt{GL_COLOR, GL_DEPTH, GL_STENCIL}

Pixel Operations

User Defined:
- draw, erase.

Bitblt (Bit Block Transfer) and Raster Ops:
- hardware supported.
- works on rectangular regions of the buffer.
- clear, set, move, and, or, xor.

Blending:
- dissolve, composite, accumulate.
Chroma Keying

Summary:

- Graphics Libraries
- OpenGL and its Structure
- Drawing Primitives in OpenGL
- GLUT and Event-Driven Programming
- Buffers and raster operations

What's Next?

- Geometry for Graphics