Overview

- OpenGL Architecture
- OpenGL Objects and Attributes
- Event-Driven Programming
- GLUT
- Open3DGL/GLUT Sample Program
- OpenGL Buffers
- OpenGL Pixel 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/CD/MDM

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.

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
- AT&T, 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 windows-specific display libraries:

- **Window Application**: Window-specific libraries
- **GL Application**: GL-specific 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**

<table>
<thead>
<tr>
<th>Data Types</th>
<th>OpenGL Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL Data Types</td>
<td>Functions: begin with gl</td>
</tr>
<tr>
<td></td>
<td>Constants: begin with GL</td>
</tr>
<tr>
<td>Types:</td>
<td>OpenGL Naming Conventions, C/C++ Data Types</td>
</tr>
</tbody>
</table>

**OpenGL Objects**

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

**General Form:**

glBegin (object type);
glVertex... (...);
glVertex... (...);
glEnd(); // don’t forget this!

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

**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:

(func_name) (dim) (type) (argument list)

Examples:
- 2D point in GLint (int) coordinates:
  glVertex2i (200, -150);
- 3D point in GLfloat (float) coordinates:
  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:
  glVertex3dv (pt);

Images from www.opengl.org/About/Architecture.html
**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 ();
```

**Polygons**

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

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

**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 ); // a
      glVertex2i ( 0, 0 ); // b
      glVertex2i ( 1, 1 ); // c
      glEnd ();
```

**In-Class Exercise**

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

**Drawing 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;
  }
}

---

**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.
- Note Robins GLUT for Win32: Has precompiled binaries for Microsoft Windows and installation instructions!

GLUT Initialization

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

- glutInitDisplayMode ( unsigned int mode )
  - Specifies a number of options affecting general operation
    connected by "or": E.g. GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE, etc.
  - Examples of Modes:
    GLUT_RGBA: Use RGB color (most common)
    GLUT_RGB: Use RGB + A color (well 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 guarantee that you will get this)

- glutInitWindowPosition ( int x, int y )
  - Request that the window be created. (This initializes the request. The window is not created immediately.)

- 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 ( )
    - Realize 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 ) {
  glutInit ( argc, argv );
  // initialize glut and gl
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );
  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;
}
```

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 "need to be redrewn",
  - 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().

- **glutMotionFunc (void (*func)(int x, int y))**
  - Mouse motion while button pressed. (x, y) is current mouse position.
  - Many OpenGL commands require that the graphics window exists.
  - When window needs to be redrawn.
  - Call the given function when a mouse button is hit.
  - The (x, y) arguments indicate where the mouse was when key was pressed.
  - See also glutMotionFunc() for shift/control keys.

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

- **glutReshapeFunc (void (*func)(int w, int h))**
  - When the window is resized.
  - Call the given function whenever the window needs to be redrawn.
  - Must be called.

- **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) (given in Part I)**
  - Callback every msecs milliseconds (or more).
  - Best effort.
  - The value parameter is used for setting multiple alarms.
  - Value parameter is used for setting multiple alarms.

- **glutSpecialFunc (void (*func)(int value))**
  - Call this after XYZ Z key is pressed.
  - This can generate lots of events!

GLUT Callback Registration (cont)

- **glutKeyboardFunc (void (*func)(int c))**
  - Passed whenever a key is hit.
  - Case switch.
  - Use switch (c) to test.
  - Call the given function whenever a key is hit or released.

- **glutMouseFunc (void (*func)(int button, int state, int x, int y))**
  - Passed whenever a mouse button is hit.
  - 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)

Sample Program (Part II)

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

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;
    windowHeight = h;
    // may need to update the projection ...
    glutPostRedisplay ();
}
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 x, int y) {
    // called if mouse click
    switch (b) {
        case GLUT_LEFT_BUTTON:
            // left button pressed
            if (s == GLUT_DOWN) ...
            else if (s == GLUT_UP) ...
            break;
            // other button events
        ...
        case GLUT_MIDDLE_BUTTON:
            // middle button pressed
            ...
        ...
        case GLUT_RIGHT_BUTTON:
            // right button pressed
            ...
        ...
    }
}
void myKeyboard (unsigned char c, int x, int y) {
    // keyboard key hit
    switch (c) {
        case 'Q':
            // 'Q' means quit
            exit(0);
            break;
            // other keyboard events
        ...
    }
}
```
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 ( 0.10, 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)
}
```

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

- **Bitmap:** A picture element. 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
- **RGBα 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 effects such as motion blur.

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

---

**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.)

**Direct Color**: 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 α-value for transparency and blending.

---

Buffers Selection in GLUT

```c
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 a 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 );
```
Reading and Writing Buffers

- **glReadPixels**: \((x, y, width, height, format, type, *pixels)\)
- **glRasterPos2i**: \((x, y)\)
- **glDrawPixels**: \((width, height, format, type, *pixels)\)
- **glCopyPixels**: \((x, y, width, height, format, type, buffer)\)

where:

- **format**:
  - GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA,
  - GL_COLOR_INDEX, GL_DEPTH_COMPONENT, ...
- **type**:
  - GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, GL_FLOAT, ...
  - GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_SHORT_5_6_5, ...
- **buffer**:
  - 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

- Image composition: original image + key image = final image

Summary

- **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