What is OpenGL?
• Graphics rendering API
  - high-quality color images composed of geometric and image primitives
  - window system independent
  - operating system independent
  - close to hardware

Various Transformations...
• How many transformations can you fit in a graphics pipeline?
  - lots

OpenGL Geometric Pipeline
• Traditional path for geometric, vertex based primitives.

OpenGL Architecture
• Direct implementation of synthetic camera
  - Image formation by computer is analogous to image formation by camera (or human)
  - Specify viewer, objects, lights
  - Let hardware/software form image (via projection)
OpenGL as a Renderer

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
  - separate pipeline for images and geometry
  - linked through texture mapping
- Rendering depends on state
  - colors, materials, light sources, etc.

OpenGL and the Windowing System

- OpenGL is concerned only with rendering
  - Window system independent
  - No input functions
- OpenGL must interact with underlying OS and windowing system
  - Need minimal interface which may be system dependent
    - Done through additional libraries: AGL, GLX, WGL

GLU and GLUT

- GLU (OpenGL Utility Library)
  - part of OpenGL
  - NURBS, tessellators, quadric shapes, etc.
- GLUT (OpenGL Utility Toolkit)
  - portable windowing API
  - not officially part of OpenGL

OpenGL Related APIs

- Application program
- Open GL, GL, GLX, AGL, WGL
- GLUT
- X, Win32, Mac OS
- software and/or hardware

Preliminaries

- Header Files
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>

- Libraries

- Enumerated Types
  - OpenGL defines numerous types for compatibility
    - GLfloat, GLint, GLenum, etc.

GLUT Basics

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Register input callback functions
    - render
    - resize
    - input: keyboard, mouse, etc.
  - Enter event processing loop
```
void main ( int argc, char** argv ){
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutCreateWindow( argv[0] );
    init();
    glutDisplayFunc( display );
    glutReshapeFunc( resize );
    glutKeyboardFunc( key );
    glutIdleFunc( idle );
    glutMainLoop();
}
```

### OpenGL Initialization

- Set up whatever state you’re going to use

```
void init( void ) {
    glClearColor( 0.0, 0.0, 0.0, 1.0 );
    glClearDepth( 1.0 );
    /* Don’t do this for Project 1!! */
    glEnable( GL_LIGHT0 );
    glEnable( GL_LIGHTING );
    glEnable( GL_DEPTH_TEST );
}
```

### GLUT Callback Functions

- Routine to call when something happens
  - window resize or redraw
  - user input
  - animation

  *“Register” callbacks with GLUT
  
  ```
  glutDisplayFunc( display );
  glutIdleFunc( idle );
  glutKeyboardFunc( keyboard );
  ```

### Rendering Callback

- Do all of your drawing here

  ```
  glutDisplayFunc( display );
  ```

```
void display( void ){
    glClear( GL_COLOR_BUFFER_BIT );
    glBegin( GL_TRIANGLE_STRIP );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    glVertex3fv( v[2] );
    glVertex3fv( v[3] );
    glEnd();  /*MAY NEED glFlush();*/
    glutSwapBuffers();
}
```

### Idle Callbacks

- Use for animation and continuous update

  ```
  glutIdleFunc( idle );
  ```

```
void idle( void ){
    t += dt;
    glutPostRedisplay();
}
```

### User Input Callbacks

- Process user input

  ```
  glutKeyboardFunc( keyboard );
  ```

```
void keyboard( char key, int x, int y ){
    switch( key ) {
    case 'q' : case 'Q' :
        exit( EXIT_SUCCESS );
        break;
    case 'r' : case 'R' :
        rotate = GL_TRUE;
    break;
    }
}
```
• Geometric Primitives
• Managing OpenGL State
• OpenGL Buffers

OpenGL Geometric Primitives
• All GL geometry is specified by vertices

OpenGL Command Formats
\[ Vertex \quad 3fv \]

Specifying Geometric Primitives
• Primitives are specified using
  \[ gl\text{Begin}( \text{primType} ); \]
  \[ gl\text{End}(); \]
  \noindent \textit{primType} determines how vertices are combined

OpenGL Color Models
• RGBA or Color Index
Language Bindings

- C standard
- C++ OK but OpenGL not object oriented
- Fortran binding exists
- Java is getting there

Interactive Computer Graphics

- Real power of computer graphics
- Display is altered in response to User input
- Helps user
  - Understand data
  - Perceive trends
  - Visualize real/imaginary objects
- Manage program - GUI
- Feedback - key ingredient!
  - Response letting user know how input was interpreted

Graphical User Interface (GUI)

- Tasks:
  - Dragging - continuous movement of graphical object according to input
  - Rubberbanding - one "end" fixed, the other wherever cursor currently is
  - Inking - trail of line segments drawn according to input movement
  - Constraints
  - Menu
- Kinds of feedback?

Basic Interaction Tasks

- Positioning
  - Spatial: Moving Cursor
  - Linguistic: Specifying X, Y values
- Selecting
- Entering Text
- Entering Numeric Quantities
  - Dials, Sliders

3D Interactive Graphics

- How to handle transition for
  - Positioning
  - Selection
  - Rotation
  - 2D to 3D Device Mapping
- Possibilities
  - Multiple Views
  - Hierarchy Traversal
  - Direct Input

Event Handling

- Versus Polling
- Event
  - Transition in control state of system
- Event Record
  - Record of system activity
    - Data structure
      - Which event
      - Data corresponding to event
      - Stored in Event Queue
Event Queue

- Managed by OS
- Keeps track of sequence
- Resources for process to deal with event
- Events are posted to it
- Front of queue passed to process that owns it
  - May be based on location

Event Loop

- Main program
- Contains Event Handler that invokes callback function that has been registered for the event that is on the front of the Event Queue

Event Handling Program

- Initialization functions
- Action functions
- Callback functions
- Main event loop
  - Invokes event handler
  - Determines which callback, if any
  - Passes control to that event handler
  - When done, control back to event handler
- Direct control of program has been removed from the program itself and moved to the USER!!!!

Events

<table>
<thead>
<tr>
<th>Events</th>
<th>Examples</th>
<th>In OpenGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>Moving</td>
<td>glutReshapeFunc</td>
</tr>
<tr>
<td></td>
<td>Resizing</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>idle</td>
<td>glutIdleFunc</td>
</tr>
<tr>
<td></td>
<td>timer</td>
<td>glutTimerFunc</td>
</tr>
<tr>
<td>Software</td>
<td>What to draw</td>
<td>glutDisplayFunc</td>
</tr>
</tbody>
</table>

Glut Menu Creation

typedef enum{ OPTION_1, OPTION_2} menu_entries;
void menu_selected(int entry);
void mouse_clicked(int button,int state,int x,int y);

void init() {
  ...
  glutCreateMenu(menu_selected);
  glutAddMenuEntry("Option 1",OPTION_1);
  glutAddMenuEntry("Option 2",OPTION_2);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
  glutMouseFunc(mouse_clicked);
  ...
}
Coordinate Systems

- Raster display and drawing system coordinates may not always be identical:

![Image](image_url)

Figure 5.1 Some common variations of chapter figures.

Coordinate Systems

- OpenGL uses "right side up" coordinates:
  - (0,0) is at bottom left
  - Coordinates increase to the right and up
- Dimensions set in call to glutInitWindowSize()

OpenGL State

- OpenGL programs run in an environment controlled by the OpenGL state:
  - Primarily, characteristics - line width, current color, etc.
- State variables maintain their values until changed
- Part of the state controls the way in which OpenGL calls cause pixels to be written
- Simplest way to set this up:

```c
glMatrixMode(GL_PROJECTION);
gLoadIdentity();
gluOrtho2D(0, 640.0, 0, 480.0);
```

Drawing in OpenGL

- Can draw more complex figures as combinations of simpler figures:

```c
void hardwiredHouse(void) {
  glBegin(GL_LINE_LOOP);
  glVertex2i(40, 40);
  glVertex2i(40, 90);
  glVertex2i(70, 120);
  glVertex2i(100, 90);
  glVertex2i(100, 40);
  glEnd();
  glBegin(GL_LINE_STRIP);
  glVertex2i(50, 100);
  glVertex2i(50, 120);
  glVertex2i(60, 120);
  glVertex2i(60, 110);
  glEnd();
  // draw the door
  // draw the window
}
```

Displaying Line Widths

- To change the line width, use:

```c
gLineWidth(GLfloat width);
```
- Width must be greater than 0.0 and default is 1.0.
- Line rendering is affected by antialiasing mode.

```c
gLineWidth(4);
```

Displaying Line Styles

- To create a line pattern use:

```c
gLineStipple(GLint factor, Glushort pattern);
```
- The pattern is a 16 bit series of 0s and 1s, beginning with the low-order bit of the pattern.
- Each bit of pattern is stretched out by a multiple of factor.

```c
gLineStipple(1, 0x3F07);
```

Displaying Line Styles

- To display a bitmapped font use:
  \texttt{glutBitmapCharacter(\texttt{void *font, int character});}

- The character is displayed at the current raster position.
  - Use \texttt{glRasterPos*()} to reposition cursor before drawing.
    
    \begin{verbatim}
    glRasterPos2i(100, 100);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'B');
    \end{verbatim}

- Supported bitmap fonts include:
  - \texttt{GLUT_BITMAP_8_BY_13}
  - \texttt{GLUT_BITMAP_9_BY_15}
  - \texttt{GLUT_BITMAP_TIMES_ROMAN_10}
  - \texttt{GLUT_BITMAP_TIMES_ROMAN_24}
  - \texttt{GLUT_BITMAP_HELVETICA_10}
  - \texttt{GLUT_BITMAP_HELVETICA_12}
  - \texttt{GLUT_BITMAP_HELVETICA_18}

Displaying Text

- To display a bitmapped font use:
  \texttt{glutBitmapCharacter(\texttt{void *font, int character});}

- The character is displayed at the current raster position.
  - Use \texttt{glRasterPos*()} to reposition cursor before drawing.
    
    \begin{verbatim}
    glRasterPos2i(100, 100);
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'B');
    \end{verbatim}

- Supported bitmap fonts include:
  - \texttt{GLUT_BITMAP_8_BY_13}
  - \texttt{GLUT_BITMAP_9_BY_15}
  - \texttt{GLUT_BITMAP_TIMES_ROMAN_10}
  - \texttt{GLUT_BITMAP_TIMES_ROMAN_24}
  - \texttt{GLUT_BITMAP_HELVETICA_10}
  - \texttt{GLUT_BITMAP_HELVETICA_12}
  - \texttt{GLUT_BITMAP_HELVETICA_18}

Polygons

- OpenGL supports some built-in polygon routines:
  \texttt{glRecti(GLInt x1, GLint y1, GLint x2, GLint y2);}

- Draws \textit{aligned} rectangle with opposite corners (x1,y1) and (x2,y2)
  - Sides are aligned with the coordinate axes

\begin{figure}[h]
  \centering
  \includegraphics[width=0.5\textwidth]{rect.png}
  \caption{Two aligned rectangles filled with colors.}
  \end{figure}

Polygons

- Can also draw as a series of vertices:
  \begin{verbatim}
  glBegin(GL_POLYGON);
  glVertex2f(x0, y0);
  glVertex2f(x1, y1);
  …
  glVertex2f(xn, yn);
  glEnd();
  \end{verbatim}

Polygons

- A polygon is \textbf{convex} if a line connecting any two points of the polygon lies entirely within it.

\begin{figure}[h]
  \centering
  \includegraphics[width=0.5\textwidth]{convex.png}
  \caption{Convex and nonconvex polygons.}
  \end{figure}

Filled Polygons

- Convex polygons can be automatically filled with a pattern.

\begin{figure}[h]
  \centering
  \includegraphics[width=0.5\textwidth]{filled.png}
  \caption{Several filled convex polygons.}
  \end{figure}
Displaying Filled Polygons

- To fill a convex polygon use:

  ```c
  glPolygonStipple(const GLuint *mask);
  ```

- The pattern is a 32x32 (128 bytes) bitmap interpreted as a mask of 0's and 1's.

- For each byte, the most significant bit is first.

  ```c
  glEnable (GL_POLYGON_STIPPLE);
  glPolygonStipple (pattern);
  glDisable(GL_POLYGON_STIPPLE);
  ```

```c
GLubyte pattern[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,
  0x04, 0x60, 0x06, 0x20, 0x04, 0x30, 0x0C, 0x20,
  0x04, 0x18, 0x18, 0x20, 0x04, 0x0C, 0x30, 0x20,
  0x04, 0x06, 0x60, 0x20, 0x44, 0x03, 0xC0, 0x22,
  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,
  0x66, 0x01, 0x80, 0x66, 0x33, 0x01, 0x80, 0xCC,
  0x19, 0x81, 0x81, 0x98, 0x0C, 0xC1, 0x83, 0x30,
  0x07, 0xe1, 0x87, 0xe0, 0x03, 0x3f, 0xfc, 0xc0,
  0x03, 0x31, 0x8c, 0xc0, 0x03, 0x33, 0xcc, 0xc0,
  0x64, 0x26, 0x60, 0x0c, 0xcc, 0x33, 0x30, 0x18,
  0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0c, 0x08,
  0x10, 0x18, 0x18, 0x08, 0x10, 0x00, 0x00, 0x08};
```

Mouse Callbacks

- Mouse presses and releases can be captured by:

  ```c
  glutMouseFunc(myMouse); // registered in main()
  ```

  ```c
  void myMouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
      ...
    }
  }
  ```

  Likewise, mouse movements once a button is pressed can be captured by:

  ```c
  glutMouseFunc(myMovedMouse); // registered in main()
  ```

  Remember, the coordinate system in GLUT requires y values to be inverted (by the current height of the screen).

Keyboard Callbacks

- Keyboard interaction is captured by:

  ```c
  glutKeyboardFunc(myKeyboard); // registered in main()
  ```

  ```c
  void myKeyboard(unsigned char key, int x, int y) {
    switch (key) {
      case 'a':
        // do something
        break;
    }
  }
  ```

Examples

- Lots of opengl examples:
  - `/usr/local/pub/ncs/graphics/OpenGL`
  - `/usr/local/glut/progs`
  - `/usr/local/glut/test`