OpenGL Drawing

- OpenGL drawing usually done in display function
- Display function is called once when program starts
- Recall that first, register callback in main( ) function
  ```c
  glutDisplayFunc( display );
  ```
- Then, implement display function
  ```c
  void display( void )
  {
    // put drawing stuff here
    glBegin( GL_LINES );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    glEnd();
  }
  ```

Basic Drawing Primitives

- Draw points, lines, polylines, polygons
- Primitives are specified using format:
  ```c
  glBegin(primType)
  // define your primitives here
  glEnd();
  
  primType: GL_POINTS, GL_LINES, GL_POLYGON...
  ```

Basic Drawing Primitives: Example

- Example: to draw three dots:
  ```c
  glBegin(GL_POINTS)
  glVertex2i(100, 50)
  glVertex2i(100, 130)
  glVertex2i(150, 130)
  glEnd();
  ```
**glBegin( ) Parameters**

- **GL_POINTS** - dots
- **GL_LINES** - lines, in pairs
- **GL_LINE_STRIP** - polylines
- **GL_LINE_LOOP** - closed loop
- **GL_TRIANGLES** - triangles, three vertices
- **GL_QUADS** - quad, four vertices
- **GL_POLYGON** - convex filled polygon

**OpenGL Command Format**

<table>
<thead>
<tr>
<th>Library</th>
<th>Command</th>
<th>Number of Arguments</th>
<th>Type of Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>gl</td>
<td>Vertex</td>
<td>2 – (x, y)</td>
<td>b = byte</td>
</tr>
<tr>
<td>glu</td>
<td>Color</td>
<td>3 – (x, y, z)</td>
<td>ub = unsigned byte</td>
</tr>
<tr>
<td>glut</td>
<td>Flush</td>
<td>4 – (x, y, z, w) or (r, g, b, a)</td>
<td>s = short, us = unsigned short, i = int, ul = unsigned int, f = float, d = double, * = wildcard</td>
</tr>
</tbody>
</table>
Some OpenGL Commands

- `glVertex2i()`: x, y vertex position
- `glColor3f()`: RGB color
- `glRecti()`: aligned rectangle
- `glClearColor`: clear color in RGB
- `glClear()`: clears screen
- `glFlush()`: forces image drawing

OpenGL Data Types

<table>
<thead>
<tr>
<th>C++</th>
<th>OpenGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed char</td>
<td>GLuint</td>
</tr>
<tr>
<td>Short</td>
<td>GLShort</td>
</tr>
<tr>
<td>Int</td>
<td>GLint</td>
</tr>
<tr>
<td>Float</td>
<td>GLFloat</td>
</tr>
<tr>
<td>Double</td>
<td>GLDouble</td>
</tr>
<tr>
<td>Signed char</td>
<td>GLubyte</td>
</tr>
<tr>
<td>Signed short</td>
<td>GLushort</td>
</tr>
<tr>
<td>Signed int</td>
<td>GLuint</td>
</tr>
</tbody>
</table>

Mouse Interaction

- Declare prototype
  - `myMouse(int button, int state, int x, int y)`
  - `myMovedMouse`
- Register callbacks:
  - `glutMouseFunc(myMouse)`: when mouse button pressed
  - `glutMotionFunc(myMovedMouse)`: when mouse moves
- Button returned values:
  - GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
- State returned values:
  - GLUT_UP, GLUT_DOWN
- X,Y returned values:
  - Coordinates of mouse location

Keyboard Interaction

- Declare prototype
  - `myKeyboard(unsigned int key, int x, int y)`
- Register callback:
  - `glutKeyboardFunc(myKeyboard)`: when keyboard is pressed
- Key values:
  - ASCII value of key pressed
- X,Y values:
  - Coordinates of mouse location
- Large `switch` statement to check which key
Example: Keyboard Callback

- How to use keyboard to control program?
- 1. register callback in main() function
  ```c
  glutKeyboardFunc( myKeyboard );
  ```
- 2. implement keyboard function
  ```c
  void myKeyboard(char key, int x, int y )
  {
    // put keyboard stuff here
    switch(key){   // check which key
      case 'f':    // do stuff
        break;
    }
  }
  ```

OpenGL State

- OpenGL tracks states
  + Drawing color
  + Point size
- Rendered objects appearance based on current state
- State variable remains active till changed

miniGL: What?

- Object-oriented wrapper in C++
- Allows you to choose to either:
  - Call pure OpenGL call or
  - Call your own OpenGL algorithm implementations
- First few projects will use pure OpenGL option
- Later, you will learn the algorithms and implement some OpenGL calls
- Google note: there exists another miniGL, OpenGL port to PDAs

miniGL: How?

- Can run executables (gear, bounce, your code) using two options:
  - `-openGL` option simply calls pure OpenGL call
  - `-cs4731GL` option calls your code
- In beginning, both options are set up to call pure OpenGL call
- Later, you will replace some parts, which are called by `-cs4731GL` option (*gulp!!*)
- Example:
  - In beginning: `-openGL`, `mgl.mglRotate` method calls `glRotate`
  - Later: `-cs4731GL`, `mgl.mglRotate` will call `home_grown_glRotate`
**miniGL: How?**

- **miniGL advantage:** debugging, can test same code using both pure calls and your code
- **How?** When coding your functions, can always use -openGL option to debug
- **Design:**
  - Encapsulate OpenGL calls in class called minigl
  - Encapsulate GLUT calls in class called miniglut
  - Use instance of minigl class called mgl
  - Use instance of miniglut class called mglut
- **Examples:**
  - `glRotate` → `mgl.mglRotate`
  - `glutMouseFunc` → `mglut.mglutMouseFunc`

**miniGL: Who?**

- **miniGL written initially by Mark Stevens**
- Stevens, previously CS professor, taught this class before
- miniGL extended by Emmanuel Agu
- Some students (Paul Tessier, Tony Andrade, etc) submitted bug fixes, corrections, etc
- miniGL mostly stable, been used in this course 3 or 4 previous times

---

**Homework 1**

- On class website
- Goal: to get you going, work out platform issues!!
- Get OpenGL and GLUT
- Set up your programming environment
- Compile miniGL code
- Examples:
  - Read sections of Hill book
  - Convert examples to miniGL format
  - Derive new HW1 class (from cs4731app class)
  - Modify miniGL draw function
  - Type in missing function(s)
  - Compile and run

---

**Homework 1**

- If you called OpenGL calls directly, it would work
- Don’t make openGL calls directly
- Always make call to miniGL
- For example:
  - Do: `mgl.mglBegin(minigl::MGL_QUADS)`
  - Don’t: `glBegin(GL_QUADS)`
- miniGL calls either call openGL calls or cs4731app (your home-grown functions).
- **Due on Friday, Sept. 5, 2003**
References

- Hill, chapter 2