2D Drawing in FLTK

See Scribble example and FLTK ch5

20 Feb 2009
CMPT166
Dr. Sean Ho
Trinity Western University

Subclassing FLTK widgets

- The basic FLTK widgets provide plenty of functionality
  - Callbacks for button press, input values, etc.
  - Programmable from Fluid
- But you can extend their functionality by writing your own subclasses of FLTK widgets
  - **class ScribbleBox : public Fl_Box { ....**
- In Fluid, insert a Box, and in the C++ tab, specify the 'Class' as our subclass ScribbleBox

Drawing: Override draw()

- The code to draw an FLTK widget is in draw()
- Subclass a widget, e.g., Fl_Box and override:
  - **ScribbleBox::draw() {**
    - **Fl_Box::draw();**
    - **// do your drawing here**
- First thing to do should be to call the superclass version of draw()
  - Draws the regular widget as though not subclassed: frame, label, etc.

Setting colour and line style

- **#include <FL/fl_draw.H>**

- Colo(u)rs:
  - Get the current pen colour: fl_color()
  - Set the colour: fl_color(Fl_Color)
  - Use named colours: FL_BLACK
    - **List in <FL/Enumerations.H>**
  - Or specify RGB triple:
    - **fl_rgb_color(128, 128, 255)**
- Line dashes and thickness:
    - fl_line_style(FL_SOLID, 2)

Window coordinate system

- Drawing in FLTK widgets is in a pixel-based coordinate system: units are screen pixels
- Origin is at the top-left corner of the window (not the widget!)
- Instance methods x(), y() provide the coordinates of the top-left corner of the widget
- w(), h() provide the dimensions of the widget
- You can draw outside your widget!

Drawing: fast shapes

- Point (single pixel): fl_point(int x, int y)
- Line (uses line style): fl_line(x1, y1, x2, y2)
- Rectangular border: fl_rect(x, y, w, h)
- Filled rectangle: fl_rectf(x, y, w, h)
- Outline triangle or quadrilateral:
    - fl_loop(x, y, x1, y1, x2, y2, x3, y3)
- Filled triangle or convex quad: fl_polygon(…)
- Elliptical sections: fl_arc/pie(x, y, w, h, a1, a2)
    - Bounding box, start/end angles in degrees

Drawing text

- fl_draw(const char* txt, int x, int y)
  - Draws txt at the specified location
- fl_font(int face, int size)
  - Specify font face and size in pixels
  - Font faces: FL_HELVETICA, FL_TIMES, etc.
  - May also add (+) modifers: FL_BOLD, FL_ITALIC

Drawing: complex shapes

- List of points: fl_begin_points(), fl_end_points()
  - Specify path in between begin and end
- List of lines: fl_begin/end_line()
- Line loop: ...._loop()
- Filled polygon (must be convex): ...._polygon()
- Complex polygon: ....._complex_polygon()
  - May have several components: fl_gap()
  - May be concave
  - May have holes: wind in opposite direction

Specifying the path

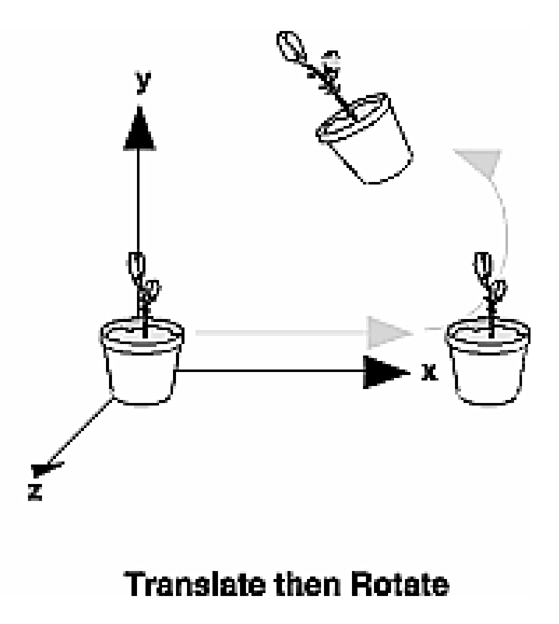
- Each of the complex objects (points, line, loop, polygon, complex_polygon) takes a path in between its begin and end. A path may have:
- Vertices: fl_vertex(float x, float y)
- Smooth "Bezier" curves:
  - fl_curve(x, y, x1, y1, x2, y2, x3, y3)
  - Interpolates through (x,y) and (x3,y3)
  - Other two are control points
- Circular arc: fl_arc(x, y, r, a1, a2)
- Complete circle: fl_circle(x, y, r)

Transformation matrix

- The complex drawing shapes use a transform matrix to determine where they are drawn
- Coordinate system need not be tied to screen pixels
- e.g., create object with dimensions 1.0x1.0, and have it scale to fill the widget
- Matrix stack: a way to save/restore current transform matrix
    - `fl_push_matrix(); // save old matrix`
    - `fl_pop_matrix(); // restore old matrix`

Combining transformations

- **fl_scale(float x, y=1);**
- **fl_translate(float x, y);**
- **fl_rotate(float degrees);**



Translate then Rotate

- Multiplies another transformation into the current transform matrix
- Operations are done in reverse order:
  - **fl_rotate(30.);**
  - **fl_translate(100., 0.);**
  - **fl_begin_polygon(); .....**
    - Translate is done first, then rotate!