2D Drawing in FLTK, cont.

See Scribble example and FLTK ch5

2 Mar 2009
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review: Drawing in FLTK

- Subclass an FLTK widget like Fl_Box
  - Override the draw() method
- Window coordinate system: pixel-based
- Fast shapes:
  - point, line, rect/rectf, loop/polygon, arc/pie
- Complex shapes: (fl_begin_* / fl_end_*)
  - points, line, loop, polygon
  - complex_polygon

Specifying the path

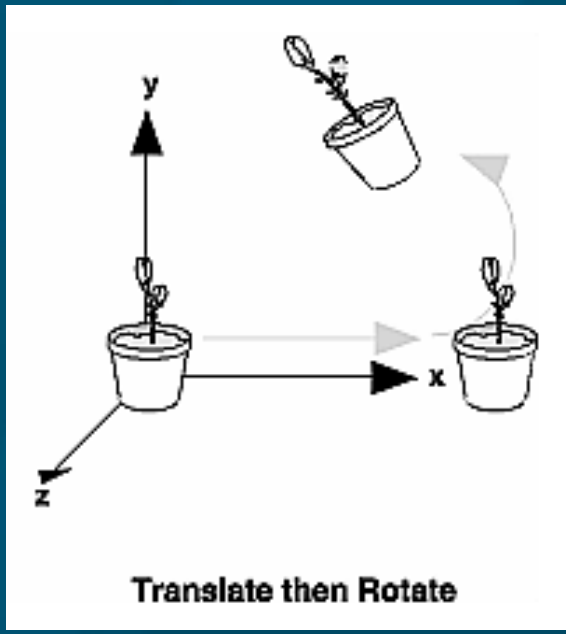
- Each of the complex objects (points, line, loop, polygon, complex_polygon) takes a path in between its begin and end. A path may have:
- Vertices: fl_vertex(float x, float y)
- Smooth "Bezier" curves:
  - fl_curve(x, y, x1, y1, x2, y2, x3, y3)
  - Interpolates through (x,y) and (x3,y3)
  - Other two 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);**

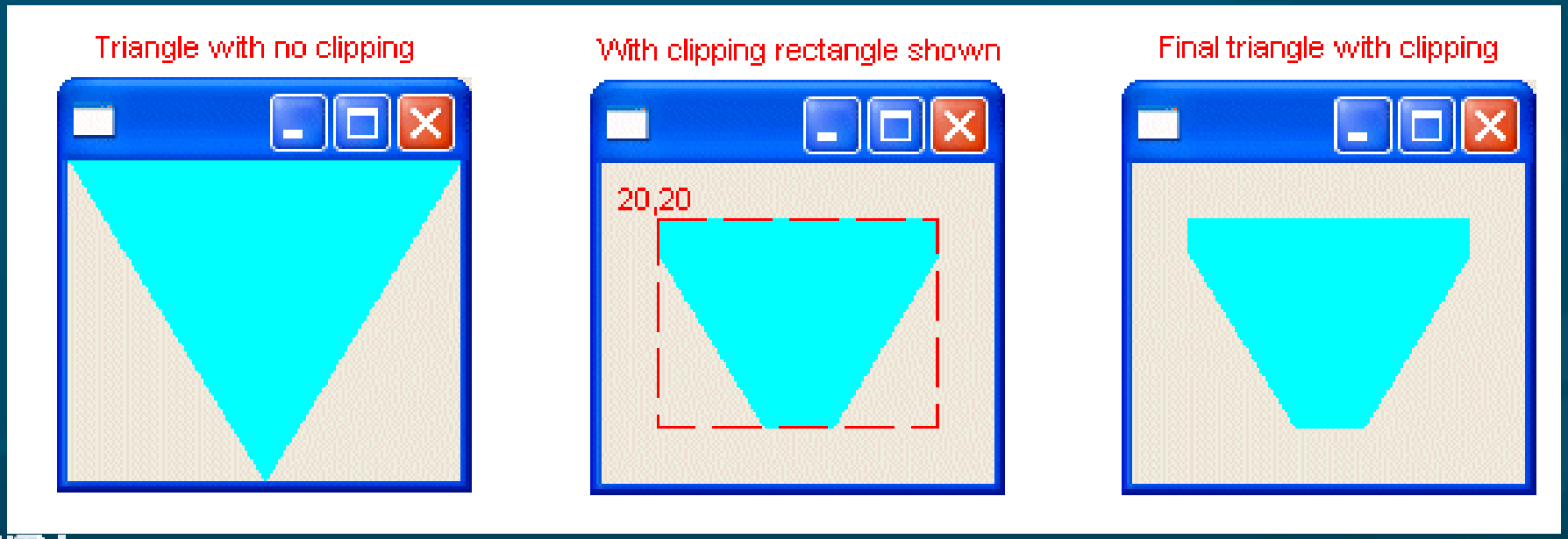


Translate then Rotate

- 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!

Clipping

- When you draw, only the portion within the current clip is actually rendered
- Usually you can draw anywhere in the window
- Use the clip to restrict drawing to only a portion of the window



FLTK clipping commands

- FLTK provides a clip stack: push/pop regions
- fl_push_clip(x, y, w, h)
  - Intersects current clip with the given rectangle and pushes it onto the stack
- fl_pop_clip()
  - Restore prevous clip
  - Every fl_push_clip() must have a corresponding fl_pop_clip()
    - **fl_push_clip(50, 50, 100, 200);**
    - **// draw stuff**
    - **fl_pop_clip();**

Drawing images

- Two ways of drawing an image using FLTK:
- Direct drawing: store image as an array of unsigned chars (bytes), uncompressed
  - RGB: e.g., a 100x200 pixel image is stored as an array of 60000 bytes
  - Faster if image changes often
- Or create an Fl_Image object
  - Faster for static: cached on display server
  - Functions to read JPG, PNG, etc. from file

Direct image drawing

- Do Fl::visual(FL_RGB); in main(), before show()ing any window
- Create your image as a flat array of uchars
  - size is width*height*3 for RGB image
- fl_draw_image(img, x, y, w, h)
  - img points to the image data
  - This is also overloaded so img may point to a function callback you write, which generates the image one line at a time

Using Fl_Image

- An Fl_Image object may be cached, reused several times easily, used as button icon, etc.
  - Use .data() to get at the pixel data
- Fl_Image is the superclass, has 3 subclasses:
  - Fl_Bitmap (black and white)
  - Fl_Pixmap (colour-mapped)
    - Each pixel of image has an index into the colourmap
  - Fl_RGB_Image (grayscale, RGB colour, with optional alpha transparency)
    - check: fl_can_do_alpha_blending()

Reading common image format

- Fl_Image also has subclasses whose constructors can load an image from file, according to its image file format:
  - GIF, JPEG, PNG, PNM, XBM, XPM
    - **#include <FL/Fl_JPEG_Image.H>**
    - **Fl_JPEG_Image img("myface.jpg");**
  - Need to link with -lfltk_images
  - See Erco's Fl_JPEG_Image example

Drawing an Fl_Image

- Methods: .copy() and .copy(w, h)
  - copy the image, optionally resizing it
- .draw(x, y, w, h, ox=0, oy=0)
  - Draws the image to the rectangle (x,y,w,h) in the window
  - Optional (ox,oy) specify where to start reading from the image: source rectangle is (ox,oy,w,h)