Thursday, November 05, 2009
3:32 PM

- Qt:
  - event-driven "paradigm"
  - for comparison, our typical programs in class are "procedural":

```
main()
{
    statement 1
    statement 2
    ;
}
```

flow of control: statement-by-statement, flow by any to end

- event-driven, instead is conceptually like a giant while loop:

```
while(event) {
    switch(event) // which event to id
        case ____ : process_event
        case ____ : process_event
    }
}
```

(like a ∞ loop)

- event examples
  - button press (mouse)
  - keyboard key press, release
  - menu item selection
  - → UI (user interaction)

- major difference between procedural programs is that the ∞ loop exists "elsewhere"; not in your code, but in toolkit, such as Qt (GTK, FLTK, etc.)

- your code gives up flow of control to toolkit

  e.g. in Qt `qapp->exec();` // NR

  ↓

  Qt starts event loop

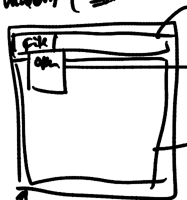
  your code sits idle, awaiting events

when an event comes along, your event handler is called (one of many that you provide)

Thursday, November 05, 2009
3:40 PM

- in my Qt examples:
  - my-texture/:
    - Makefile (top-level Makefile)
      - when you 'make' (usually just needed once), it goes into src/ & runs qmake on the .pro file that's there — qmake builds a Makefile in src/
    - src/

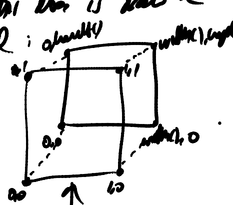
- in my-texture/src/:
  - texture.pro — config file for qmake which creates a makefile
  - glwindow.cpp — builds the window (widget)
  - gltexture.cpp



initializeGL:
- sets up GL state
- window size
- projection

paintGL:
- drawing event handler
- all that does is draw a quad: glVertex() + a texture map

redraw event needed, e.g. after resize, move, uncover



contains image (pixel) data (e.g. lena.ppm)

need a texture id to access.

only when

Qt events ≡ signals

event handlers ≡ callbacks ≡ slots