FLTK intro

Breve introduzione alla programmazione del toolkit FLTK
ogni osservazione o correzione è benvenuta

Introduzione

Il presente documento è una breve introduzione alla programmazione del toolkit FLTK (www.fltk.org). Per una trattazione completa si veda il manuale ufficiale allegato alla distribuzione (e richiamabile tramite l'help di FLUID), oppure la documentazione on-line (http://www.fltk.org/documentation.php).

Caratteristiche principali e vantaggi del FLTK

- Leggerezza e reattività del risultato finale.
- Facilità di apprendimento e velocità di sviluppo.
- Le funzioni di libreria utilizzate sono linkate staticamente, quindi gli eseguibili generati non necessitano di installazione per funzionare su una macchina diversa da quella di sviluppo. Ciononostante le dimensioni sono ridotte.
- E' multipiattaforma, Linux/Unix, Windows e MacOSX.
- Supporta la libreria OpenGL per funzioni grafiche avanzate.
- Non includendo altre funzioni, permette di concentrarsi sullo sviluppo grafico, scegliendo liberamente le altre librerie.
- Nella distribuzione sono inclusi un editor (FLUID), la documentazione necessaria (HTML) e degli utili esempi.

Studio degli esempi allegati

Una volta scaricati e compilati i sorgenti, si troveranno nella sottodirectory *test* una serie di esempi, già compilati e ovviamente completi di codice sorgente. Lo studio di tali esempi permette di imparare le tecniche di base dell'fltk.

Per studiare più chiaramente un esempio è più comodo isolarlo dalla affollata directory *test*. Ad esempio, per il codice *cube* (funzionante con la libreria OpenGL):

- Creare un nuova directory e copiarvi il sorgente *cube.cxx* dalla directory *test*.
- Commentare nel sorgente la riga

```
#include < config.h >
```

- Aprire un terminale, portarsi nella directory del sorgente e impartire il comando:

```
g++ cube.cxx ‘fltk-config --ldflags --use-gl‘ -o cube
```

E' quantomeno sorprendente il risultato ottenuto con un sorgente di 4.7Kb ed un eseguibile (statico) di 171Kb.

Nel caso di un esempio costruito con FLUID (fast_slow):

- Creare un nuova directory e copiarvi il sorgente *fast_slow.fl* dalla directory *test*.
- Lanciare *fluid* e aprire il file appena copiato.
- Selezionare dal menu di FLUID: *File->Write Code*
- Aprire un terminale, portarsi nella directory del sorgente e impartire il comando:

```
g++ fast_slow.cxx ‘fltk-config --ldflags‘ -o fast_slow
```

In questo caso il risultato è ottenuto da un sorgente di soli 1.2Kb! E' interessante notare come l'eseguibile sia ottenuto partendo dal solo file .fl.

Il comando fltk-config

Come molti toolkit anche fltk, una volta installato, mette a disposizione un comando (*fltk-config*) che permette di automatizzare la creazione dei flags per il compilatore ed il linker. Per ottenere l'elenco delle opzioni possibili digitare:

```
fltk-config --help
```

Il comando può essere utilizzato direttamente all'interno di un comando di compilazione: (attenzione: l'apice da usare è quello invertito: [altGr]['])

```
g++ elenco_dei_sorgenti `fltk-config --ldflags` -o nome_dell_eseguibile
```

oppure all'interno di un makefile:

```
LDFLAGS=`fltk-config --ldflags`
CFLAGS=`fltk-config --cflags`
```

FLUID, il visual editor di fltk

La distribuzione ufficiale di FLTK include anche un, essenziale, editor per la costruzione visuale delle interfacce.

fig1: l'interfaccia di FLUID

fig2: la "widget bin"

Fluid utilizza un semplice file di testo per salvare l'interfaccia costruita. Per convenzione si utlizza l'estensione *.fl* (che va inserita manualmente al momento del salvataggio del progetto).

Una volta terminata la costruzione del'interfaccia sarà sufficiente, dal menù *file* selezionare la voce *Write Code*. Fluid scriverà il codice (C o C++) necessario per realizzare l'interfaccia. Tale codice può essere utlizzato sia copiandolo all'interno del nostro sorgente, sia come file a sè stante, includendolo nel source tree. E' inoltre possibile editare il codice direttamente all'interno di fluid, costruendo l'applicazione direttamente nell'editor.

L'uso è intuitivo, si tengano presenti però alcune caratteristiche:

- Per iniziare la costruzione di una finestra è necessario prima inserire una nuova funzione. Il codice per costruire la finestra verrà posto all'interno della funzione specificata.
- Lasciando libero il campo *name* della funzione il nome verrà automaticamente posto a main(), in questo modo è possibile scrivere un interfaccia compilabile direttamente in fluid.
- Fluid non crea autonomamante una nuova directory per un nuovo progetto, si dovrà quindi provvedere manualmente.
- Il tasto di cancellazione è [CTRL][X]
- I nuovi elementi vengono aggiunti in modo gerarchico, quindi quando si vuole inserire un elemento assicurarsi che nella finestra principale sia evidenziata la corretta posizione.
- L'ordine degli elementi nella finestra principale rispecchia l'ordine nel codice, quindi non è ininfluente. Per spostarli si utilizzano i tasti [F2] e [F3].
- Nella versione 1.1.5 il salvataggio e scrittura del codice associati al comando *shell execute* non funzionano come dovrebbero. Infatti salvano i file nella home dell'utente e non nella cartella di progetto. E' quindi meglio salvare a mano prima di ogni compilazione e ricorrere ad un terminale per i comandi.

Esempio di utilizzo di FLUID

In questo esempio verrà spiegato, passo per passo, come realizzare una semplice finestra di countdown. Si apprenderanno le tecniche elementari di utlizzo di FLUID. (download: countdown.fl)

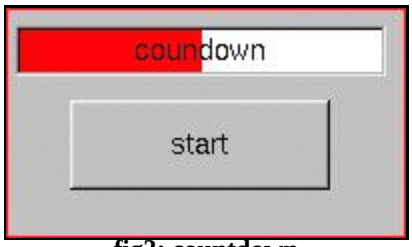

fig3: countdown

Si comincierà con il creare l'interfaccia tramite Fluid, senza preoccuparsi, inizialmente, del codice.

- Lanciare fluid, scegliere *file->new*
- Selezionare (o creare) una directory vuota, assegnare un nome al nuovo progetto (scrivendolo di seguito al path nella finestra di selezione). Nel seguito assumeremo che il nome scelto sia *countdown.fl*
- Selezionare dal menù : *new->code->function*, nella successiva finestra cancellare il nome della funzione (verrà posto a *main*)
- Selezionare la funzione *main* ed inserire una nuova finestra (menù: *new->group->window*)
- Se non è ancora visibile al finestra dei componenti (*Bin*) selezionare *edit->widget bin on/off* (oppure [Alt][b])
- Aggiungere alla finestra un bottone premendo il tasto *button* della widget bin (o selezionando *new->buttons->button*) e successivamente cliccando sulla finestra.
- Selezionare *file->write code*
- Selezionare *shell->execute command* e nella edit box inserire:

```
g++ countdown.cxx 'fltk-config --ldflags' -o countdown && countdown
```

A questo punto l'eseguibile *countdown* è pronto nella directory di progetto. Chiaramente, non avendo aggiunto codice, l'interfaccia funziona, ma non fa nulla.

Per aggiungere una funzione di callback (eseguita alla pressione del pulsante):

- Dalla finestra delle proprietà del bottone, nella scheda C++, inserire nella casella *Callback* il nome della funzione chiamata: *button_cb* (questa è la funzione che verrà chiamata alla pressione del bottone).
- Selezionare *new->code->function*, inserire nella casella nome *button_cb(Fl_Widget*,void*)* e nella casella return type *void*
- Selezionare *new->code->code* ed aggiungere il codice eseguito dalla funzione:

```
exit(0);
```

- Per compilare selezionare *shell->execute again*. A questo punto la pressione del bottone causa la chiusura della finestra.

Aggiungiamo anche una progress bar che funzioni da conto alla rovescia:

- Nella finestra aggiungere anche un oggetto fi tipo *Progress*, nella scheda *C++* delle proprietà inserire il nome *countdown* (questo è il nome dato all'oggetto all'interno del codice).
- Selezionare la funzione main e aggiungere un blocco di codice (menù: *new->code->code*). Nella edit box si può inserire il codice che verrà eseguito dopo l'istanza dell finestra (assegna l'intervallo e il valore attuale alla progress bar):

```
countdown->maximum(10000);
countdown->minimum(0);
countdown->value(10000);
```

- Facendo doppio click sul codice della funzione *button_cb* modificare il contenuto in:

```
int i;
for (i=10000;i>0;i--) {
      countdown->value(i);
      Fl::check();
}
exit(0);
```

  Nota: la funzione *Fl::check* serve a rinfrescare l'interfaccia durante il ciclo.

A questo punto il progetto dovrebbe apparire come in **fig1**. La progress bar realizza un countdown prima della chiusura della finestra.
