CSE 564: Computer Graphics

Lab 1 Setup

Klaus Mueller

Computer Science Department

Stony Brook University

Assignment

Build a user interface that supports

- file read and write
- menu buttons
- various image processing routines you learned about in class
- exit

Image processing routines:

- per-pixel operations: brighten, color manipulations
- linear filtering: blur, sharpen, edge detect, emboss
- non-linear filtering: median smooth, bilateral filter
- resampling: scale, rotate, special effects
- artistry: painterly, non-photorealistic effects, mood light

User Interface

There are many choices, determined by:

- flexibility
- appearance
- support
- learning curve
- most have visual GUI (Graphical user Interface) builders

FLTK

- public domain (http://www.fltk.org)
- C/C++
- OpenGL for graphics rendering (we will teach with OpenGL)

.NET

- commercial, Microsoft
- supports C/C++, C#
- DirectX for graphics rendering

Others: Qt,

FLTK

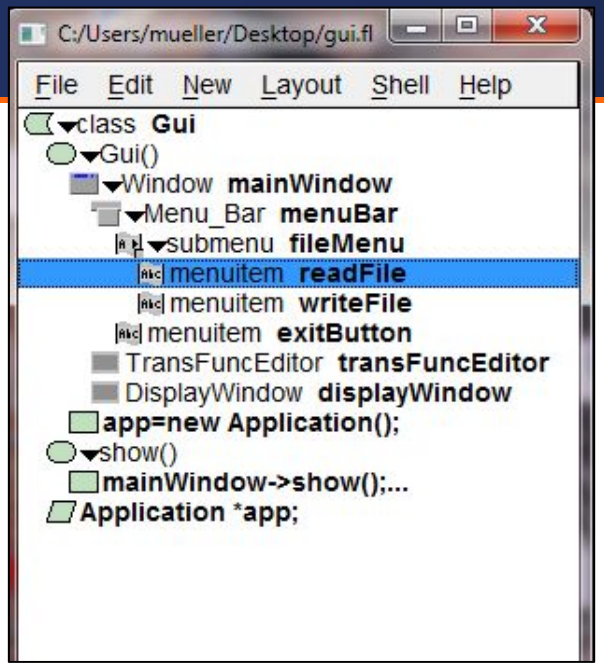
Download from http://www.fltk.org

- 1.x version (not 2.x or 3.x) is recommended
- need to build with Visual Studio C/C++
- various libraries (debug and release)
- lots of online documentation

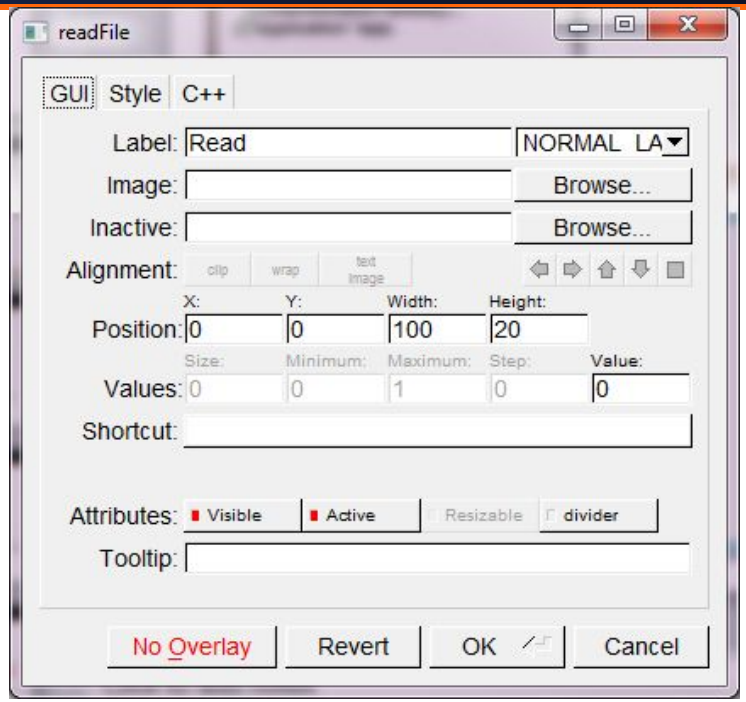
GUI editor FLUID

- allows construction of GUI
- writes .cpp files for VC projects
- need to link buttons with C-functions

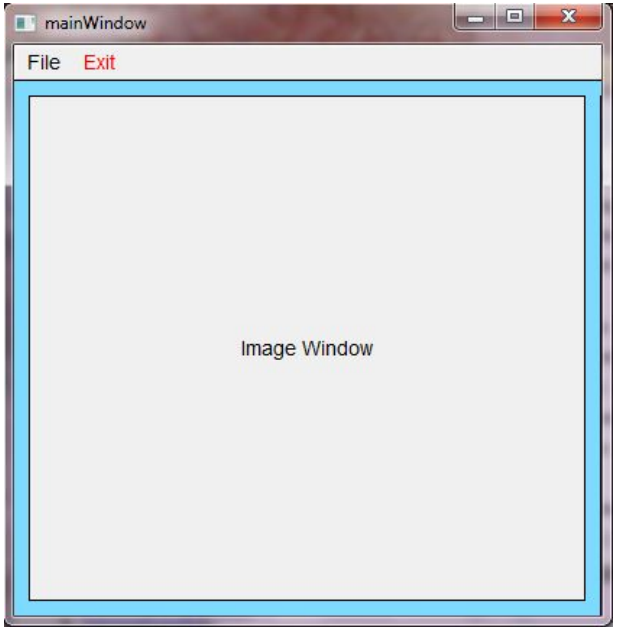
FLUID



FLUID interface



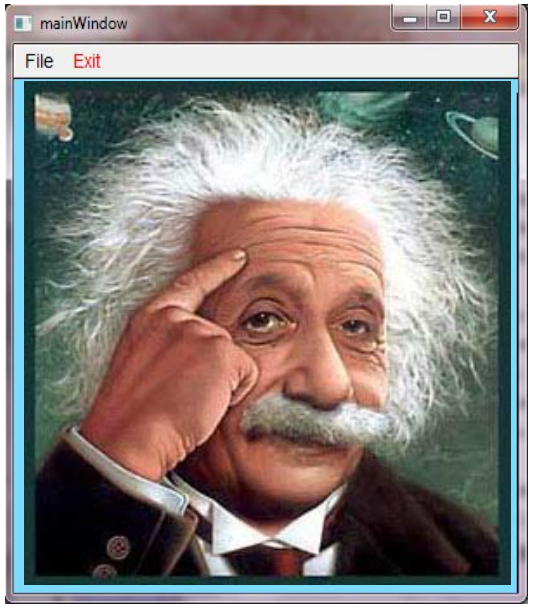
ReadFile button designer



GUI under design

Main.cpp

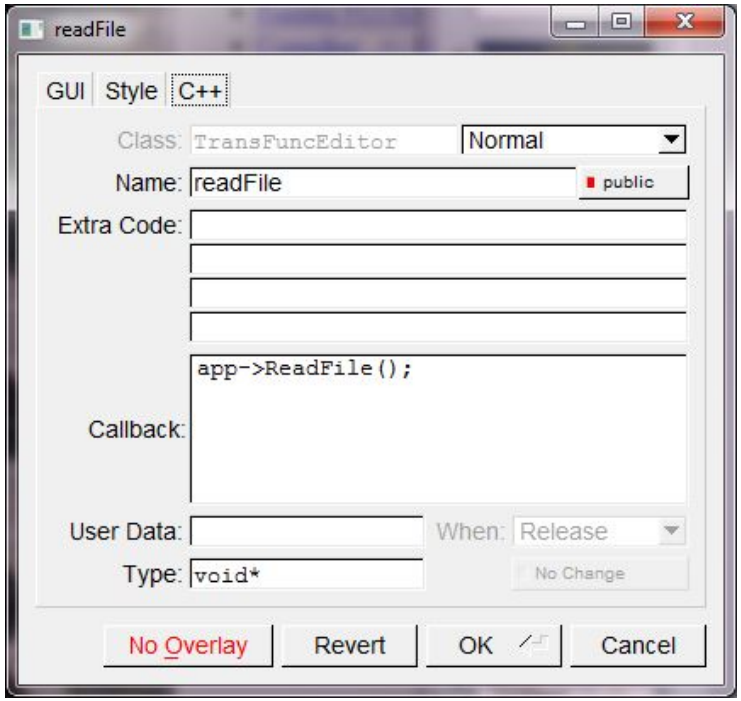
```
#include <FL/Fl.H>
#include "Gui.h“ // built by fltk

// main routine: builds the GUI, shows it, then goes into an endless event-driven loop
Gui *gui;
main(int argc, char *argv[])
{
   gui=new Gui;    // makes the GUI
   gui->show();      // shows the GUI
   return Fl::run();  // returns and waits for events (mouse-click)
}
```



Gui.cpp

File built by fltk

- can choose any other name when saving with FLUID
- in this case you get gui.cpp and gui.h
- maps all events to function calls



Function call app->readFile()

app is a C++ class in application.cpp

Image Formats

Read/write images

- PPM is a really easy format to read and write
- can use irfanview to convert any image to ppm

PPM format

- first line: P6 // code
- second line: nx ny // # of pixels in x and y direction
- third line: 255 // number of levels
- sometimes there is a comment line starting with #
- next lines: (binary) pixel values in x-axis order, r g b r g b r g b …
- example:

```
P6
# created by irfanview
100 256
255
lots of 8-bit numbers
```

OpenGL

To show the image you need to know some OpenGL (or DirectX)

Brief introduction

- more about this later when we talk about 3D

Best is to make a separate class DisplayWindow()

- call it with gui->displayWindow->redraw();
- constructor method:

```
DisplayWindow::DisplayWindow(int x,int y,int w,int h,const char *l)
      : Fl_Gl_Window(x,y,w,h,l)
{
   // clear window
  glClearColor(0.0,0.0,0.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}
```

Display Window

```
void DisplayWindow::draw() {
    if (!valid()) {
      glLoadIdentity(); glViewport(0,0,w(),h()); gluOrtho2D(0,w(),0,h());  // set up viewport and transform
      make_current();
 }
// clear window first
   glClearColor(0.0,0.0,0.0,0.0); // white
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // color and depth
   if(curImage.nx==0)  return;

// display the image in the center of the window
// h() returns the height of the window, w() returns the width
   glPixelStorei(GL_UNPACK_ALIGNMENT,1);  // byte-alignment
   glRasterPos2i((w()-curImage.nx)/2,(h()-curImage.ny)/2);  // makes sure the image appears centered
   glDrawPixels(curImage.nx,curImage.ny,GL_RGB,GL_UNSIGNED_BYTE,curImage.data);  // draw image
}
```