Impressionist Help Session
Overview

• Skeleton code
• OpenGL
• FLTK
• Requirements
  – Brushes
  – Alpha blending
  – Filter kernel
• Debugging hints
• Git tutorial (those who are familiar don’t have to stay)
Skeleton Code

**impressionistDoc**
The data/information about brushes, gradient, image, edge map, ...

**impressionistUI**
Contains the user interface. Windows, menus, size slider, brush choice dialog, etc.

**impBrush**

- **pointBrush**
- **circleBrush**
- **scatteredPointBrush**
- **scatteredLineBrush**
- **scatteredCircleBrush**
- **lineBrush**
**impressionistDoc**
Handles all of the document-related items, like loading/saving, etc

**impressionistUI**
Handles the UI components, such as getting values from sliders, setting up the window, etc

**paintView**
Handles the drawing side of the window the user paints on (right side). A lot of event handling is done here (good place to look for examples)

**originalView**
Handles the original image side of the window (left side)

**impBrush**
The virtual class all brushes are derived from

**pointBrush**
An example brush that draws points
OpenGL

• Great environment for PC 2d/3d graphics applications
• Extremely well documented
• We will be using it throughout the quarter
• This project uses the basics of OpenGL
  • Although you’re welcome to learn more on your own, the focus of the project is on 2d image manipulation
How OpenGL Works

• OpenGL draws primitives – lines, vertexes, or polygons – subject to many selectable modes

• It can be modeled as a state machine
  • Once a mode is selected, it stays there until turned off

• It is procedural – commands are executed in the order they’re specified
Drawing a Primitive

// Let’s draw a filled triangle!

// first, set your color

setColor3f(red, green, blue);

// tell OpenGL to begin drawing

begin( GL_POLYGON );

// specify vertices A, B, and C.

vertex2d(Ax, Ay);

vertex2d(Bx, By);

vertex2d(Cx, Cy);

// close the OpenGL block

end();

// Force OpenGL to draw what you specified now

flush();
FLTK

- Stands for Fast Light ToolKit (pronounced fulltick)
- Cross-platform C++ GUI toolkit
- Completely event-driven (via callbacks)
- We’ll use fltk 1.3.2
- impressionistUI.cpp has a lot of widget examples
Brushes

• Let’s make a triangle brush! (this will not count towards extra credit)
• Make a copy of pointBrush.h/cpp and rename to triangleBrush.h/cpp and add to the impressionist project
• Go through the code and change all pointBrush labels to triangleBrush
Brushes, cont’d

• Go to impBrush.h and add BRUSH_TRIANGLE to the enum
• Open impressionistDoc.cpp, add triangleBrush.h to the includes. Scroll down a bit, and add triangleBrush to the selectable brushes.
• Go to impressionistUI.cpp and add the triangle brush to the brush menu
Modify the BrushMove method to draw a triangle instead of a point in triangleBrush.cpp

```cpp
int size = pDoc->getSize();
int Ax,Ay,Bx,By,Cx,Cy;
Ax = target.x - (.5*size);
Bx = target.x + (.5*size);
Cx = target.x;
Ay = target.y - (.5*size);
By = target.y - (.5*size);
Cy = target.y + (.5*size);
glBegin( GL_POLYGON );
    SetColor( source );
    glVertex2d( Ax, Ay );
    glVertex2d( Bx, By );
    glVertex2d( Cx, Cy );
glEnd();
```
Edge detection & gradients

• The gradient is a vector that points in the direction of maximum increase of $f$

$$\nabla f = \frac{\partial f}{\partial x} \hat{x} + \frac{\partial f}{\partial y} \hat{y}.$$  

$$\theta = \text{atan2} \left( \frac{\partial f}{\partial y}, \frac{\partial f}{\partial x} \right).$$

• Use the sobel operator
Alpha Blending

\[ F_{new} = \alpha C + (1-\alpha) F_{old} \]

Make sure you call `glBlendFunc` outside the `glBegin/glEnd` block.
Filters

• Remember how filter kernels are applied to an image
  – Look at the sample solution. How does it apply a filter?
  – What could go wrong?
  – What cases do you need to handle?

• We will be looking closely at your filter kernel
Use Gimp/Photoshop to see filters
Debugging

One thing that might help is to be checking for errors after each call. When it seems like nothing is happening, OpenGL is often returning an error message somewhere along the line. The begin-end block is a good possibility, and if that's the problem there will be an error code returned.

GLenum error_flag;

error_flag = glGetError();

if (error_flag != GL_NO_ERROR) {
    printf("Error: %s (%i) in %s.\n", gluErrorString(error_flag),
            error_flag, "method name");
}
Git

This quarter: git
Previous quarters: svn

Resources
Basics for this course:

Official Documentation:

git --help <command>