Project 1: Impressionist Help Session

What we'll be going over:

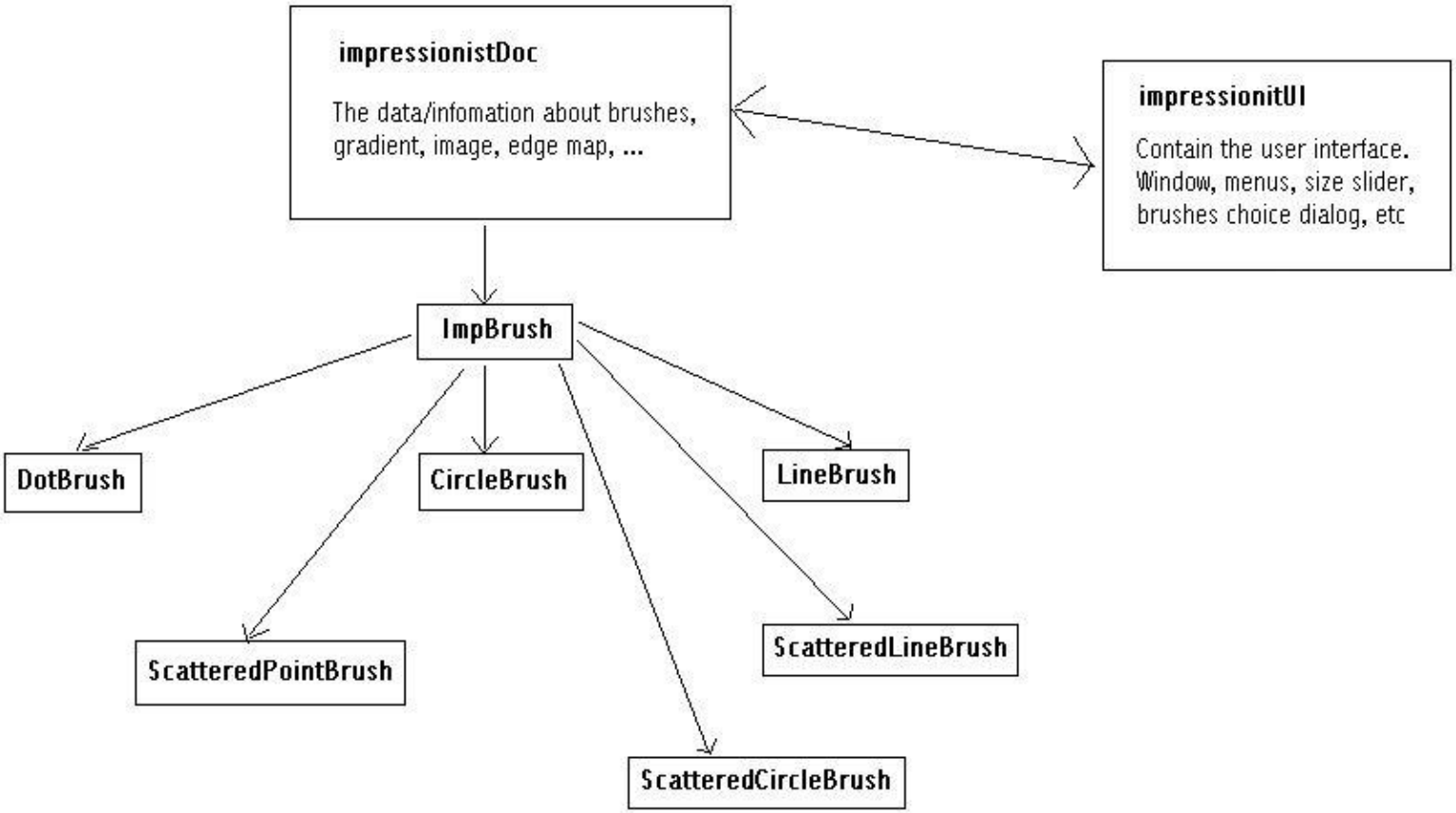
- SVN: How to use it.
- The Skeleton Code: Live and Uncut!
- What on earth is OpenGL (and why doesn't it work)?
- Basic FLTK
- Hint #1: How to make a new brush.
- Committing the changes you made.
- Hint #2+: Filter kernel dos and don'ts
- Basic Photoshop (artifacts, plus check your work!)
- Good References for Project 1
- How to set up MSVC .NET from home (or, the other labs)
- Q&A

SVN: The Basics

- Set up space… done for you!
- Import the code… also done for you!
- Check out a local copy

  Right-click on SVN Checkout, enter the proper repository and module and BAM! You're done!
- Make some changes (we'll see this later on in the session)
- Add any new files (we'll see this later on in the session)
- Commit your code (also will be seen later on in the session)

The Skeleton Code



The Skeleton Code cont'd

- ImpressionistDoc
  - This class handles all of the document-related stuff, like loading/saving, etc.
- ImpressionistUI
  - This class handles all of the UI stuff, such as getting values from sliders, setting up the window, etc.
- PaintView
  - This class handles drawing the side of the window the user paints on.
  - A lot of event handling is done here (so here look for examples).
- OriginalView
  - This class handles the other side of the window.
- ImpBrush
  - This is the virtual class all brushes are derived from.
- PointBrush
  - This is an example brush that draws points.

Meet your new friend (or enemy): OpenGL

- OpenGL is a great environment for PC 2d/3d graphics applications.
  - It is one among many others, such as DirectX, Glide, Allegro, etc.
- Very easy to start working with—trust me!
  - It is extremely well documented.
- We will be using it throughout the quarter.
- Project 1 uses just 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.
- The coordinate system in which it draws is transformed using function calls.
  - glRotate, and why it might be confusing (right now).
  - The matrix stack.
  - This will all be covered in detail in the modeler help session!

Drawing with openGL

- That said, how to draw an actual primitive?
  - Lets do an example: a filled triangle.

First, set your color:

```
glColor3f( red, green, blue );
```

Now, tell openGL to begin drawing:

```
glBegin( GL_POLYGON );
```

Specify vertices A, B, and C. Since we're drawing in an image, use integers.

```
  glVertex2i( Ax, Ay );
  glVertex2i( Bx, By );
  glVertex2i( Cx, Cy );
```

Close the openGL block.

```
glEnd();
```

Force openGL to draw what you specified *now*.

```
glFlush();   // don't forget this!
```

FLTK: Diet Win32

- Stands for Fast Light ToolKit.
  - A *really* handy cross-platform windowing system.
- Completely Event-driven (via callbacks).
  - The window setup code is run, and then the main loop is called. (we'll look at an example in a second)
  - All further events are handed out to callbacks.
- New version has been released (fltk 1.1.10)

FLTK Example code

- This code is taken/modified directly from fltk.org:

```
#include <put junk here>
```

This code is executed in order:

```
int main(int argc, char **argv) {
  Fl_Window *window = new Fl_Window(300,180);
  Fl_Box *box = new Fl_Box(20,40,260,100,"Hello, World!");
```

Run functions registered to Fl_Box on the box you created:

```
  box->box(FL_UP_BOX);
  box->labelsize(36);
  box->labelfont(FL_BOLD+FL_ITALIC);
  box->labeltype(FL_SHADOW_LABEL);
  window->end();
  window->show(argc, argv);
```

This is where we hand control of our program to FLTK. Anything that happens now is the result of a callback.

```
  return Fl::run(); }
```

How to Make a Brush

- Now that we've got all the background, lets make a brush!
  - Presenting. . .triangleBrush! (Note: This will NOT count towards extra credit)
- Easiest way to do this: make a copy of pointBrush.h/cpp and rename them triangleBrush.h/cpp.
- Add them to the impressionist project.
- Go through the code and change all pointBrush labels to triangleBrush.

Brushmaking, cont'd

- Now, open up ImpressionistDoc.cpp
- Add triangleBrush.h to the includes
- Scroll down a bit, and add triangleBrush to the selectable brushes. Pick a constant for it.
- Go to ImpBrush.h and add the constant for triangleBrush to the enum.
- Go to impressionistUI.cpp, and add the triangle brush to the brush menu.

Brushmaking, cont'd again

- Run Impressionist. See the triangle brush.
  - And, well, see the triangle brush make points instead of triangles.
- Open triangleBrush.cpp and go to BrushMove.
  - Here's what's there now:

```
glBegin( GL_POINTS );
        SetColor( source );

        glVertex2d( target.x, target.y );
glEnd();
```

- Triangles need 3 vertices. Lets center ours around the target point where the user clicked.
- How do we do this?

Brushmaking, cont'd again

- We do it like so:

```
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 );
    glVertex2i( Ax, Ay );
    glVertex2i( Bx, By );
    glVertex2i( Cx, Cy );
glEnd();
```

Commit your changes

- Now that we've made some changes, we should commit them back to the repository.
- We can do this by right-clicking on the folder and choosing TortoiseSVN->Add (you only need to do this if you have created a new file)
  - Make sure to check any new files (*.h,*.cpp,*.vcproj) that you may have created. In this example, it would be triangleBrush.h and triangleBrush.cpp
- Then right-click the folder again and choose SVN Commit
- Enter a message describing the changes, then hit OK

Filter Kernel Hints

- Remember how filter kernels are applied to an image.
  - Look at sample program. How does it apply a filter?
  - What could go wrong?
  - What cases should we handle?
- We will be looking closely at your filter kernel. . . Very closely

OpenGL Debugging Hints

- 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: %1s (%i) in
    %1s.\n",gluErrorString(error_flag),error_flag,"method name");
}
```

Basic Photoshop

- How to check your work. . .
  - Filter kernel in Photoshop:
    - Go to Filter->Other->Custom
  - Median filter in Photoshop:
    - Go to Filter->Noise->Median
- How to resize your artifact. . .
  - Go to Image->Image Size

Setting up FLTK from home

- Download fltk 1.1.10 from the course webpage
- Put it wherever you think is convenient (say, C:\fltk-1.1.10)
- Open up the impressionist project
- From tools, select options
- Go to the projects tab and select VC++ directories
- Under “show directories for” choose “Include files”
- Create a new directory as the fltk folder (in my case, C:\fltk-1.1.10)
- Also add the directories for the fltk\zlib, png, and jpeg (i.e. C:\fltk-1.1.10\zlib, C:\fltk-1.1.10\png, and C:\fltk-1.1.10\jpeg)
- Under “show directories for” choose “Library files”
- Create a new directory for fltk\lib (in my case, C:\fltk-1.1.10\lib)

- This is all on the website, so don’t need to copy it down

Good References

- Books around the lab!!
  - The Red/Blue OpenGL Bibles
- Class Web
  - Lots of references linked there!
- Google
- www.fltk.org
- www.opengl.org
- Your TAs

Questions. . .?

- Ask ‘em now. . .
- …Or email the staff list (csep557-staff@cs) later
- General OpenGL questions and clarifications can also be directed to the discussion board