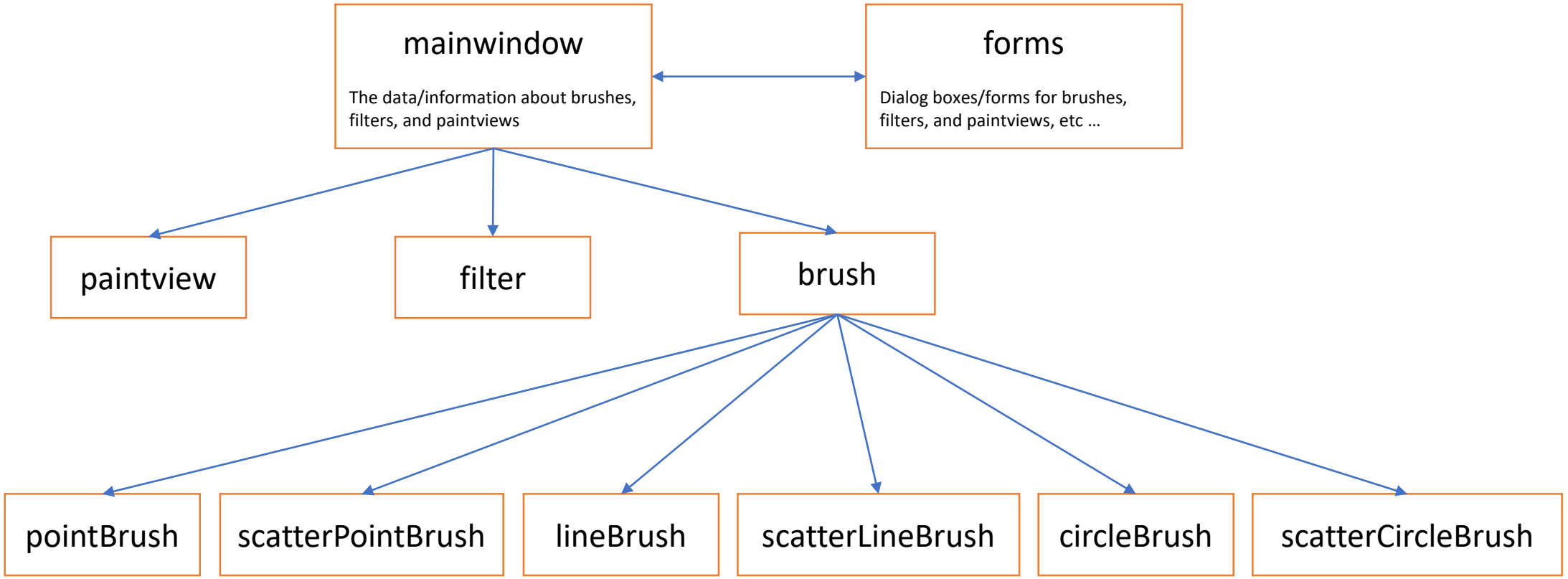
Impressionist Help Session

Overview

- Skeleton code
- OpenGL
- Qt
- Requirements
  - Brushes
  - Alpha blending
  - Filter kernel
  - Mean bilaternal filter
- Debugging hints
- Git tutorial (those who are familiar don’t have to stay)

Skeleton Code



- mainwindow

  Handles all of the document related items like loading and saving, selecting brushes, and applying filters

- forms

  Various UI components (the main window, brush & kernel dialog boxes, etc..)

- paintview

  Handles the original image side of the window (left side) and the drawing side of the window the user paints on (right side)

- brush

  The virtual class all brushes are derived from

- pointbrush

  An example brush that draws points

OpenGL

- Good(ish) environment for PC 2d/3d graphics applications
- Extremely well documented… well not really!
  - Lots of beginner tutorials online
  - www.khronos.org/opengl/wiki/
    - Keys to understanding how OpenGL works
    - But sometimes has unfinished pages
- 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 (and we encourage this), the focus of the project is on 2d image manipulation

How OpenGL Works

- OpenGL draws primitives – lines, vertices, or polygons – subject to many selectable modes
- It can be modeled as a state machine
  - Once a mode is set, 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
glColor3f( red, green, blue );

// tell OpenGL to begin drawing
glBegin(GL_POLYGON);
       // specify vertices A, B, and C.
       glVertex2d( Ax, Ay );
       glVertex2d( Bx, By );
       glVertex2d( Cx, Cy );
// close the OpenGL block
glEnd();

// Force OpenGL to draw what you specified now
glFlush();
```

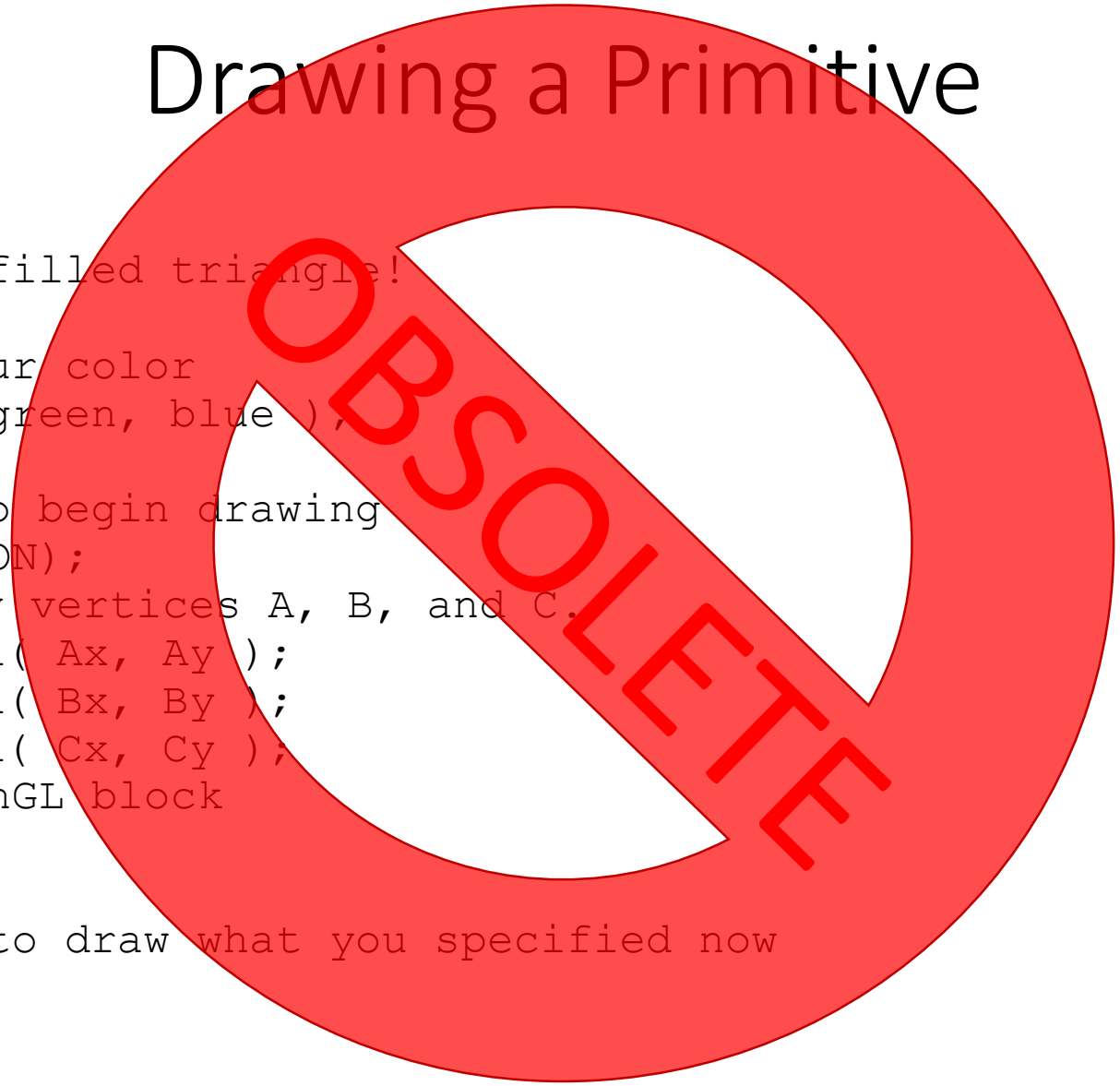
Drawing a Primitive

```
// Let's draw a filled triangle!

// first, set your color
glColor3f( red, green, blue );

// tell OpenGL to begin drawing
glBegin(GL_POLYGON);
       // specify vertices A, B, and C.
       glVertex2d( Ax, Ay );
       glVertex2d( Bx, By );
       glVertex2d( Cx, Cy );
// close the OpenGL block
glEnd();

// Force OpenGL to draw what you specified now
glFlush();
```



Drawing a Polygon

```
// Let’s draw a filled triangle!

// first, set your color
glm::vec4 color;
color.r = red;
color.g = green;
Color.b = blue;

// set the vertices
Std::vector<Glfloat> vertex = {
        Ax, Ay,
        Bx, By,
        Cx, Cy
};

// send the vertex data to the GPU buffer
glBufferData(GL_ARRAY_BUFFER, sizeof(float)*vertex.size(), vertex.data(), GL_STREAM_DRAW;

// Draw polygon
glDrawArrays(GL_TRIANGLES, 0, 3);
```

Drawing a Polygon

- A lot going on behind the scenes
- There is a lot of prep code needed to draw
  - We need to create vertex array object that records all the state needed to draw a brush, bound every time we draw
  - We need to create a vertex buffer object to hold the vertex positions and specify the format of the vertex data (`GL_LINES`, `GL_TRIANGLES`, `GL_QUADS`, …many more!)
  - We need to create a shader program (we did this for you)

Qt

- Enables developers to develop applications with intuitive user interfaces for multiple targets, faster than from scratch
  - It's a cross-platform GUI toolkit
  - We needed a windowing toolkit to handle window/rendering context creation for OpenGL since we don't want to do that ourselves
  - FLTK (what we used to use) is lightweight, but has sparse features that don't play as well with nicer, newer hardware
- Event-Driven (via callbacks as slot and signal pairings)
- We're supporting Qt 5.7, although version 5.8 is the latest and works
- QtCreator IDE – installed with Qt
- mainwindow.cpp has several widget examples

Brushes

- Let’s make a triangle brush! (this will of course NOT count towards extra credit)

- Make a copy of pointbrush.h/cpp and rename to trianglebrush.h/cpp
  - Right-click pointbrush.h/cpp -> Duplicate File…
  - Right-click pointbrush_copy.h/cpp -> Rename…
  - Rename to “trianglebrush.h/cpp”
  - They should show up as part of the impressionist project

- Go through the trianglebrush.h/cpp code and change all pointbrush labels to trianglebrush labels

Brushes, cont’d

- Go to brush.h and add Triangle to the Brushes enum class

- Open forms/brushdialog.cpp, add <brushes/trianglebrush.h> to the includes. Scroll down a bit, and add the triangle brush to the selectable brushes.

Brushes, cont’d

Modify the BrushMove method to draw a triangle instead of a point in trianglebrush.cpp

```
int size = GetSize();
std::vector<GLfloat> vertex = {
    pos.x - (size * 0.5f), pos.y + (size * 0.5f),
    pos.x + (size * 0.5f), pos.y + (size * 0.5f),
    pos.x, pos.y - (size * 0.5f)
};

glBufferData(GL_ARRAY_BUFFER, sizeof(float) * vertex.size(), vertex.data(), GL_STREAM_DRAW);

glDrawArrays(GL_TRIANGLES, 0, 3);
```

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

- $\mathrm{F}_{new} = \alpha C + (1 - \alpha)\mathrm{F}_{old}$

If $\alpha = 0.5, C = \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \end{bmatrix}, \mathrm{F}_{old} = \begin{bmatrix} 255 \\ 0 \\ 0 \\ 128 \end{bmatrix}$

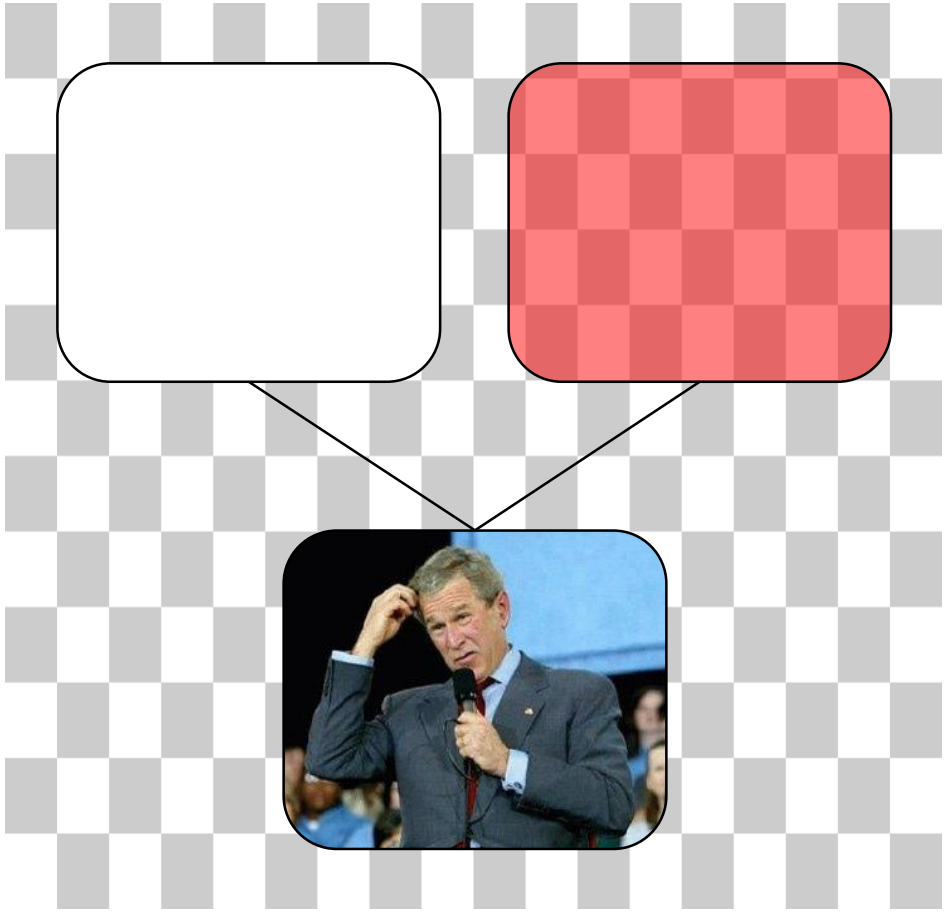
Then $\mathrm{F}_{new} = \begin{bmatrix} ? \\ ? \\ ? \\ ? \end{bmatrix}$

Alpha Blending

- $\mathrm{F}_{new} = \alpha C + (1 - \alpha)\mathrm{F}_{old}$

If $\alpha = 0.5, C = \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \end{bmatrix}, \mathrm{F}_{old} = \begin{bmatrix} 255 \\ 0 \\ 0 \\ 128 \end{bmatrix}$

Then $\mathrm{F}_{new} = \begin{bmatrix} ? \\ ? \\ ? \\ ? \end{bmatrix}$



Alpha Blending

- $\mathrm{F}_{new} = \alpha C + (1 - \alpha)\mathrm{F}_{old}$

If $\alpha = 0.5, C = \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \end{bmatrix}, \mathrm{F}_{old} = \begin{bmatrix} 255 \\ 0 \\ 0 \\ 128 \end{bmatrix}$

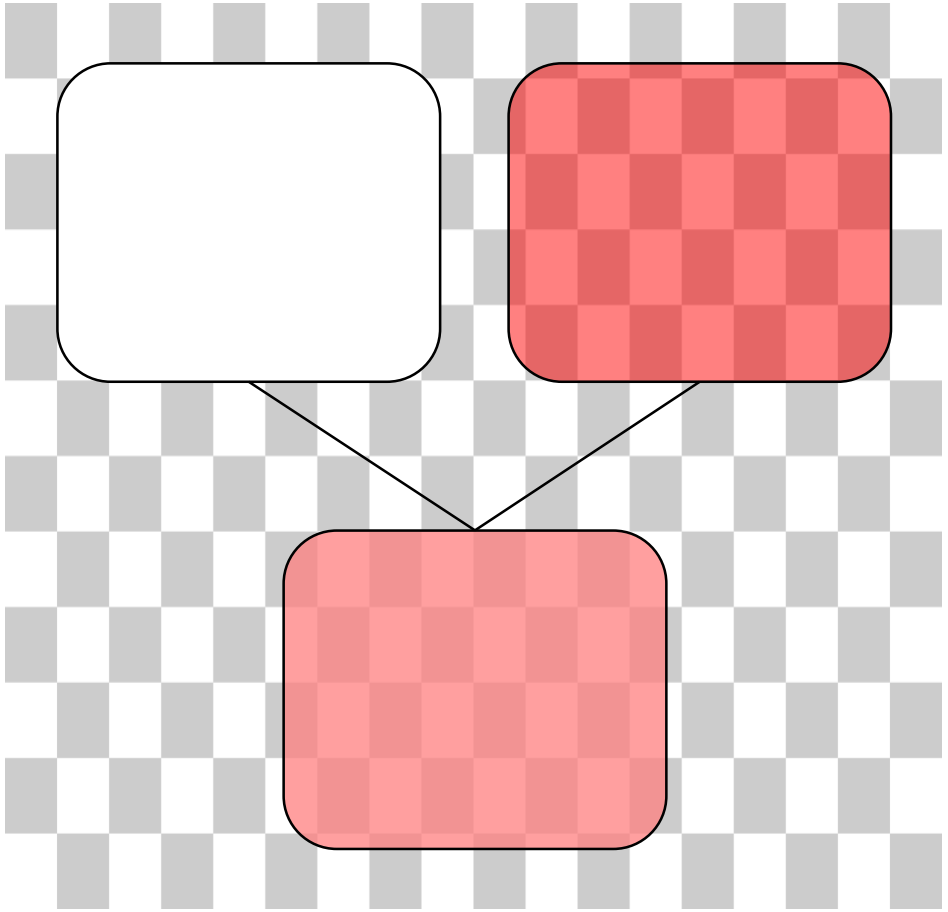
Then $\mathrm{F}_{new} = 0.5 \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \end{bmatrix} + (1 - 0.5) \begin{bmatrix} 255 \\ 0 \\ 0 \\ 128 \end{bmatrix} = \begin{bmatrix} 128 \\ 128 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 128 \\ 0 \\ 0 \\ 64 \end{bmatrix} = \begin{bmatrix} 255 \\ 128 \\ 128 \\ 192 \end{bmatrix}$

Alpha Blending

- $\mathrm{F}_{new} = \alpha C + (1 - \alpha)\mathrm{F}_{old}$

If $\alpha = 0.5, C = \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \end{bmatrix}, \mathrm{F}_{old} = \begin{bmatrix} 255 \\ 0 \\ 0 \\ 128 \end{bmatrix}$

Then $\mathrm{F}_{new} = \begin{bmatrix} 255 \\ 128 \\ 128 \\ 192 \end{bmatrix}$

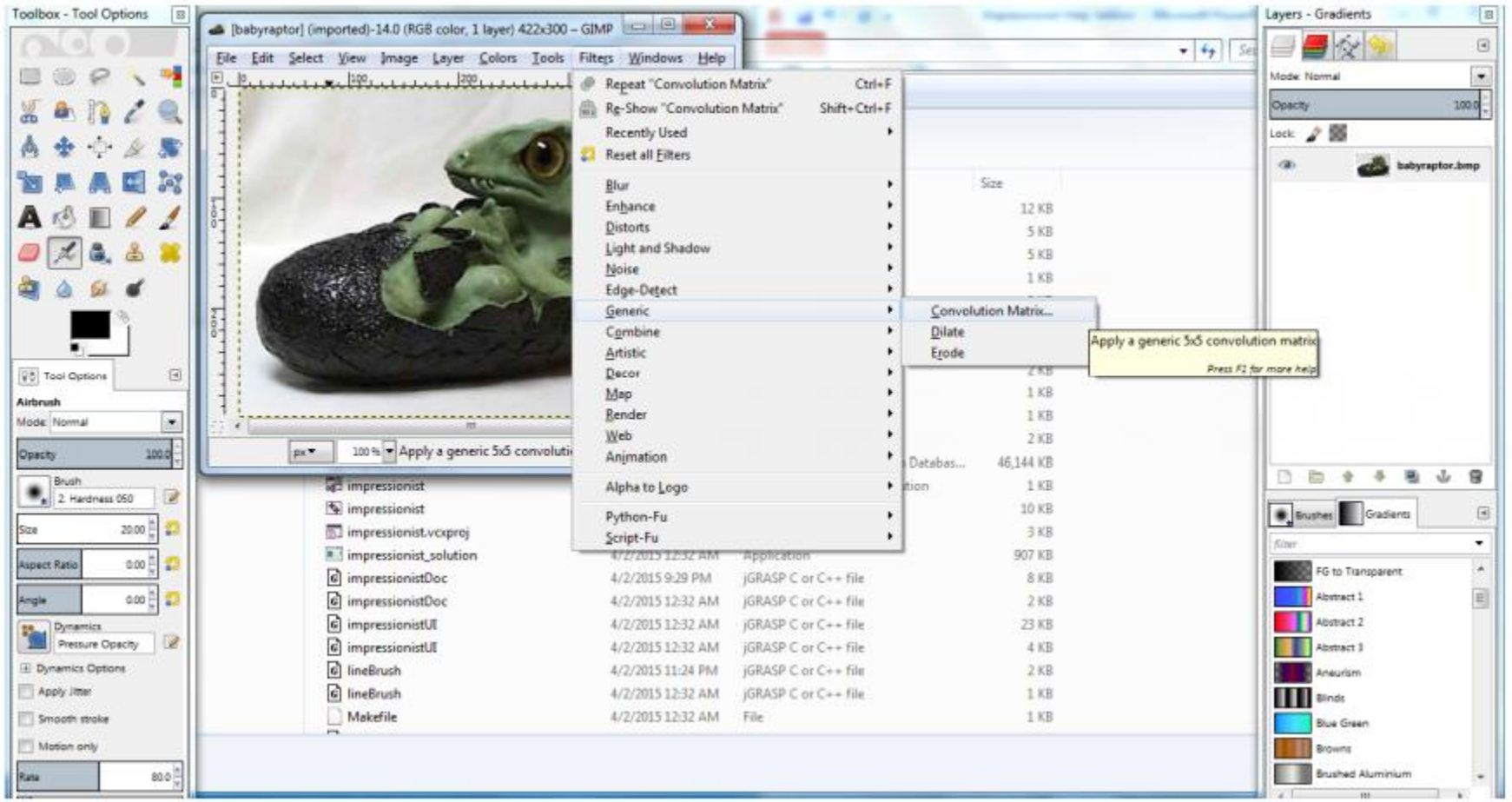


Filters

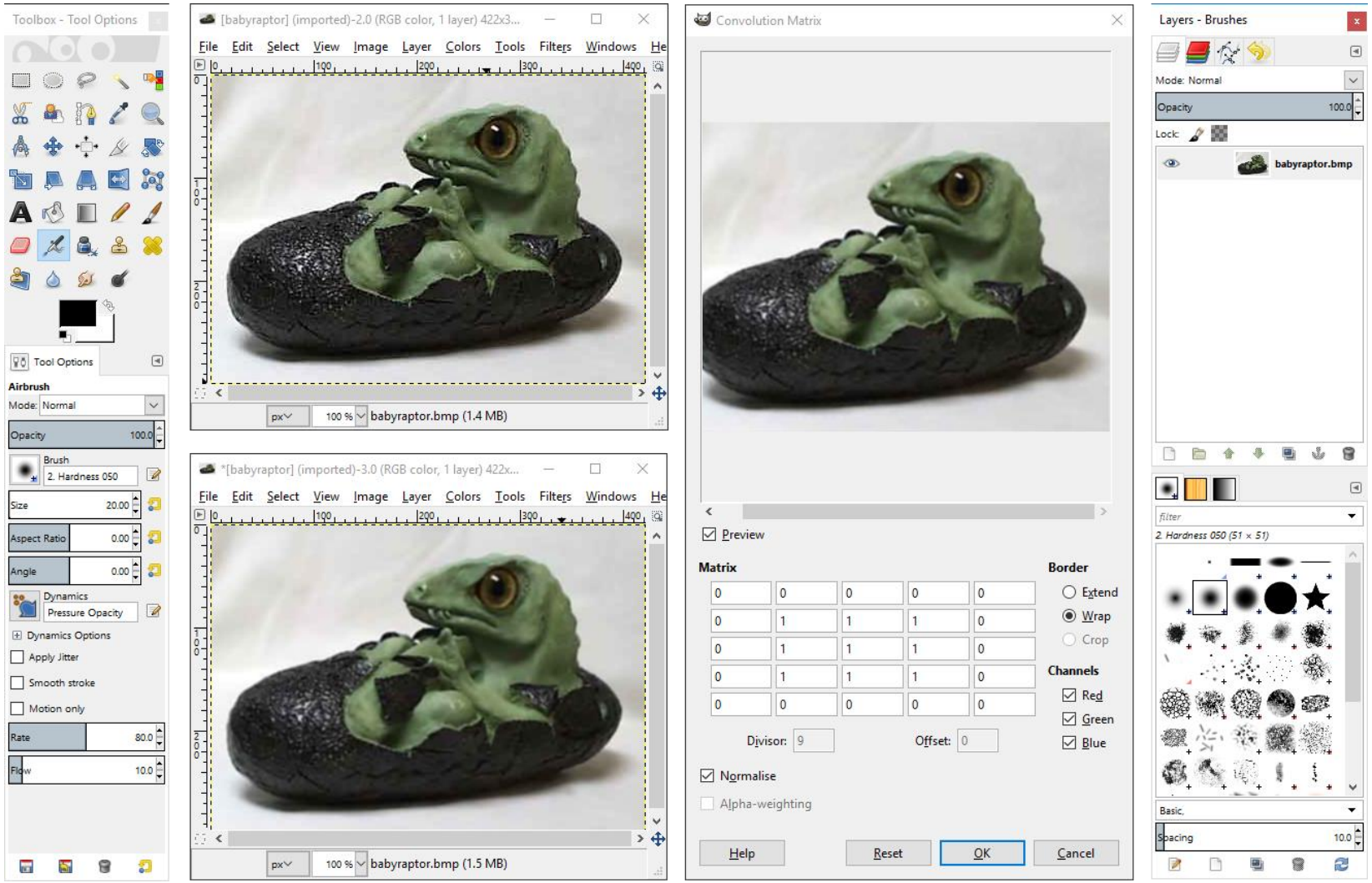
- Remember how filter kernels are applied to an image
  - Look at the sample solution. How does it apply a filter?
  - What could go wrong?
  - What cases do you need to handle?

- We will be looking closely at your filter kernel

Use GIMP/Photoshop to see filters in action



3x3 Mean Box Filter



Debugging

- Debugging in Qt
  - Use Qt's built-in debugger (works just like VS, Eclipse, or just about any IDE you've used).
  - Print out debugging info
    - #include <QDebug>
    - Use `qDebug()` when you want to display information

```
qDebug() << "debugging info: " << debugInfo;
```

  - Rebuild the project
    - Clean → Make → Build the Project
- Debugging OpenGL
  - It might help to check for errors after each call. When it seems like nothing is happening, OpenGL is often returning an error message somewhere along the line.
    - #include <glinclude.h>
    - Use `GLCheckError();`

Git

- Resources
  - Basics for this course:

    https://courses.cs.washington.edu/courses/cse457/17sp/src/help.php

  - Official documentation:

    https://git-scm.com/book/en/v2

    `git --help <command>`

Git, cont’d

- Starting
  - navigate to the directory you want to work in and run

    $ git clone git@gitlab.cs.washington.edu:cse457-17sp-impressionist/YOUR_REPO.git impressionist
  - This clones your repository into a working directory named “impressionist”

- Working
  - You will want to periodically check your code in, either to avoid disaster or to rollback broken code to an earlier working version, run

    $ git add --all

    $ git commit -m "added a triangle brush“

    $ git push
  - If you made any changes remotely, run

    $ git pull

Git, cont'd

- Finished, Code turn-in
  - Build your executable in **Release Mode**
  - Be sure to have everything properly committed and pushed to your GitLab repository first

    $ git status

    ✓ On branch master

    ✓ Your branch is up-to-date with "origin/master"

    ✓ Nothing to commit, working directory clean
  - Tag it

    $ git tag SUBMIT

    $ git push –tags
  - Clone your tagged repo into a **SEPARATE** directory and test running the program