HELP SESSION

IMPRESSIONIST

OUTLINE

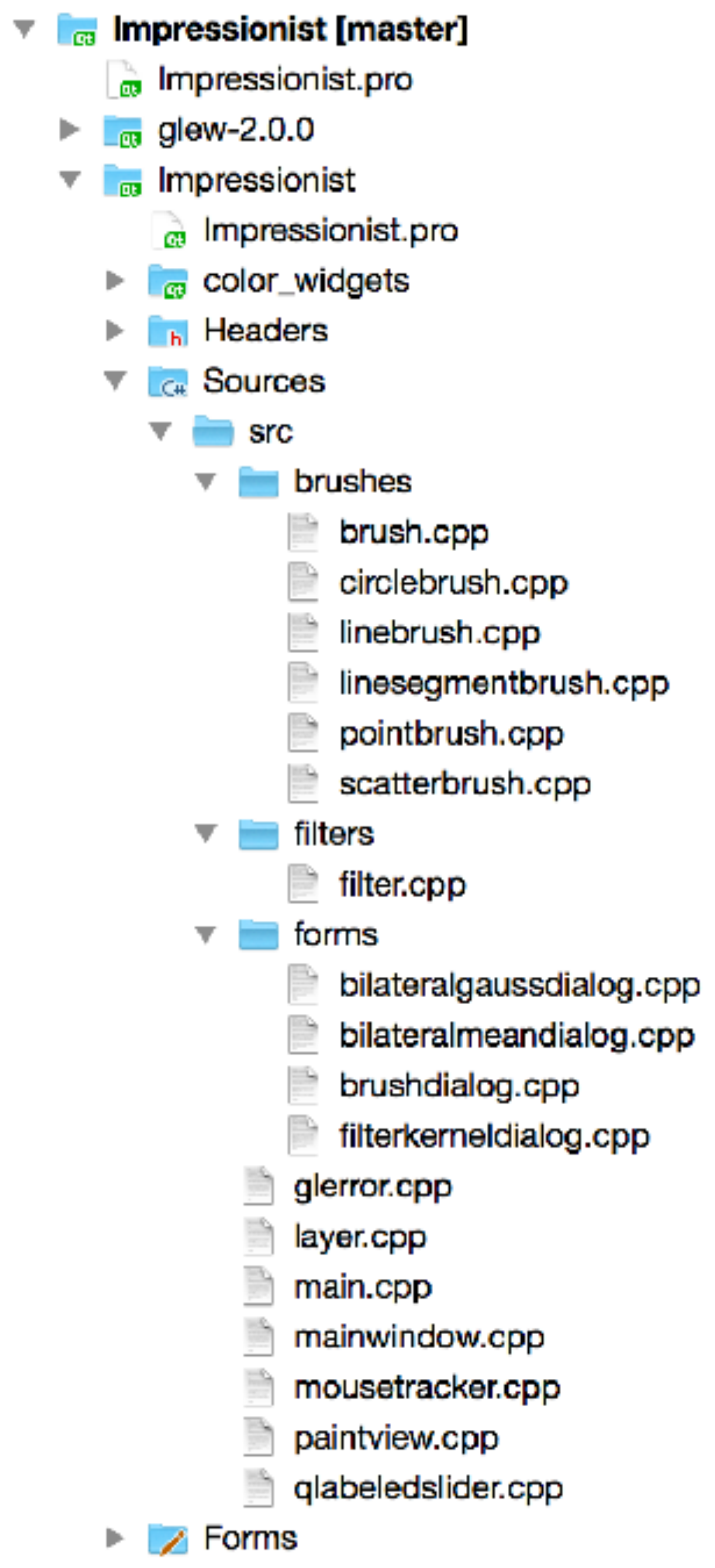
- Skeleton Code
- OpenGL
- Qt
  - Debugging Hints
- Project requirements
  - Brushes
  - Alpha Blending
  - Filter Kernel
  - Mean Bilateral Filter
- Git Tutorial

GETTING STARTED

- Clone the Impressionist skeleton code
  - `git clone git@gitlab.cs.washington.edu:csep557-19sp-impressionist/YOUR_REPO.git impressionist`
- Install Qt Creator (if working on your own machine)
  - www.qt.io/download > Go Open Source
  - On Windows, first install the MSVC C++ compiler
    - Installing Visual Studio (not Visual Studio Code) with C++ support enabled will do this
- In Qt Creator, "Open Existing Project" and open Impressionist.pro

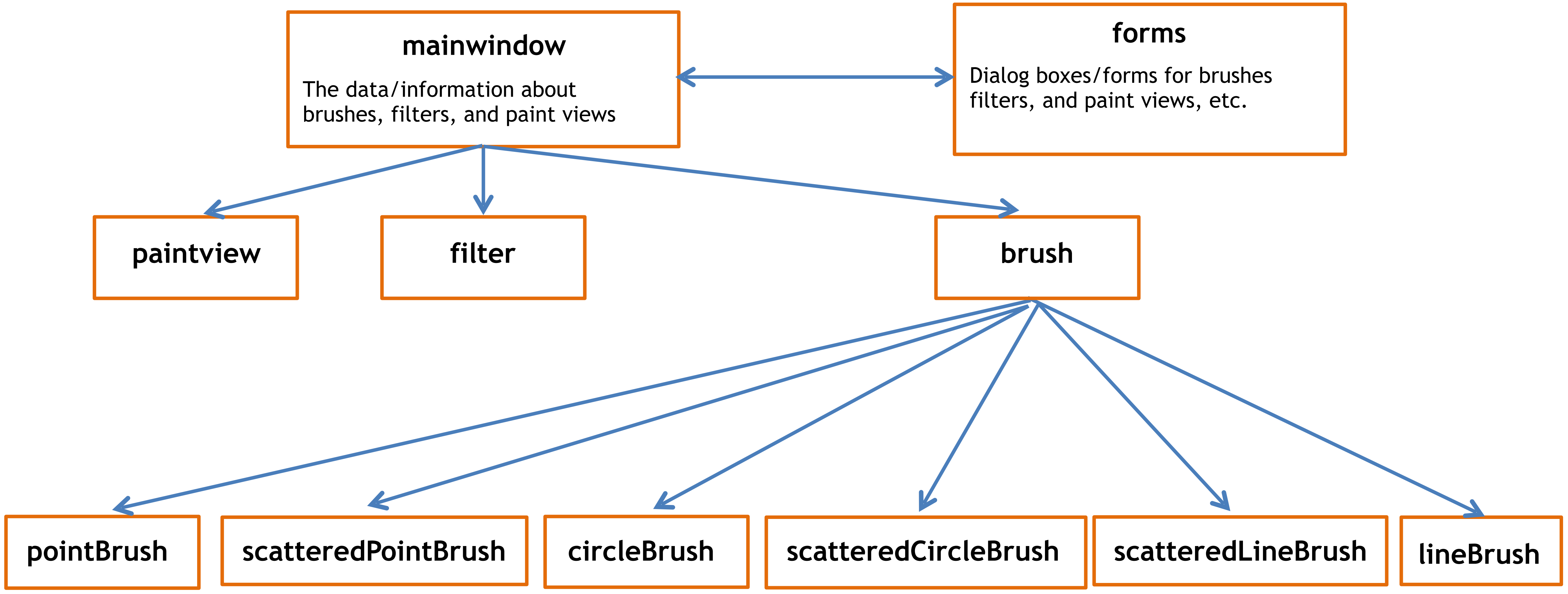
SKELETON CODE

SKELETON CODE



```
▼ Impressionist [master]
    Impressionist.pro
  ▶ glew-2.0.0
  ▼ Impressionist
      Impressionist.pro
    ▶ color_widgets
    ▶ Headers
    ▼ Sources
      ▼ src
        ▼ brushes
            brush.cpp
            circlebrush.cpp
            linebrush.cpp
            linesegmentbrush.cpp
            pointbrush.cpp
            scatterbrush.cpp
        ▼ filters
            filter.cpp
        ▼ forms
            bilateralgaussdialog.cpp
            bilateralmeandialog.cpp
            brushdialog.cpp
            filterkerneldialog.cpp
          glerror.cpp
          layer.cpp
          main.cpp
          mainwindow.cpp
          mousetracker.cpp
          paintview.cpp
          qlabeledslider.cpp
    ▶ Forms
```

SKELETON CODE



FILES

- `mainwindow.[h|cpp]`
  - 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.[h|cpp]`
  - Handles the original image side of the window (left side) and the drawing side of the window the user paints on (right side)
- `brush.[h|cpp]`
  - The virtual class all brushes are derived from
- `pointbrush.[h|cpp]`
  - An example brush that draws square points

OPENGL

- Good(ish) environment for PC 2d/3d graphics applications
- Extremely well documented… well not really!
  - Lots of beginner tutorials online (like learnopengl.com)
  - 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 this 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 are specified

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);
```

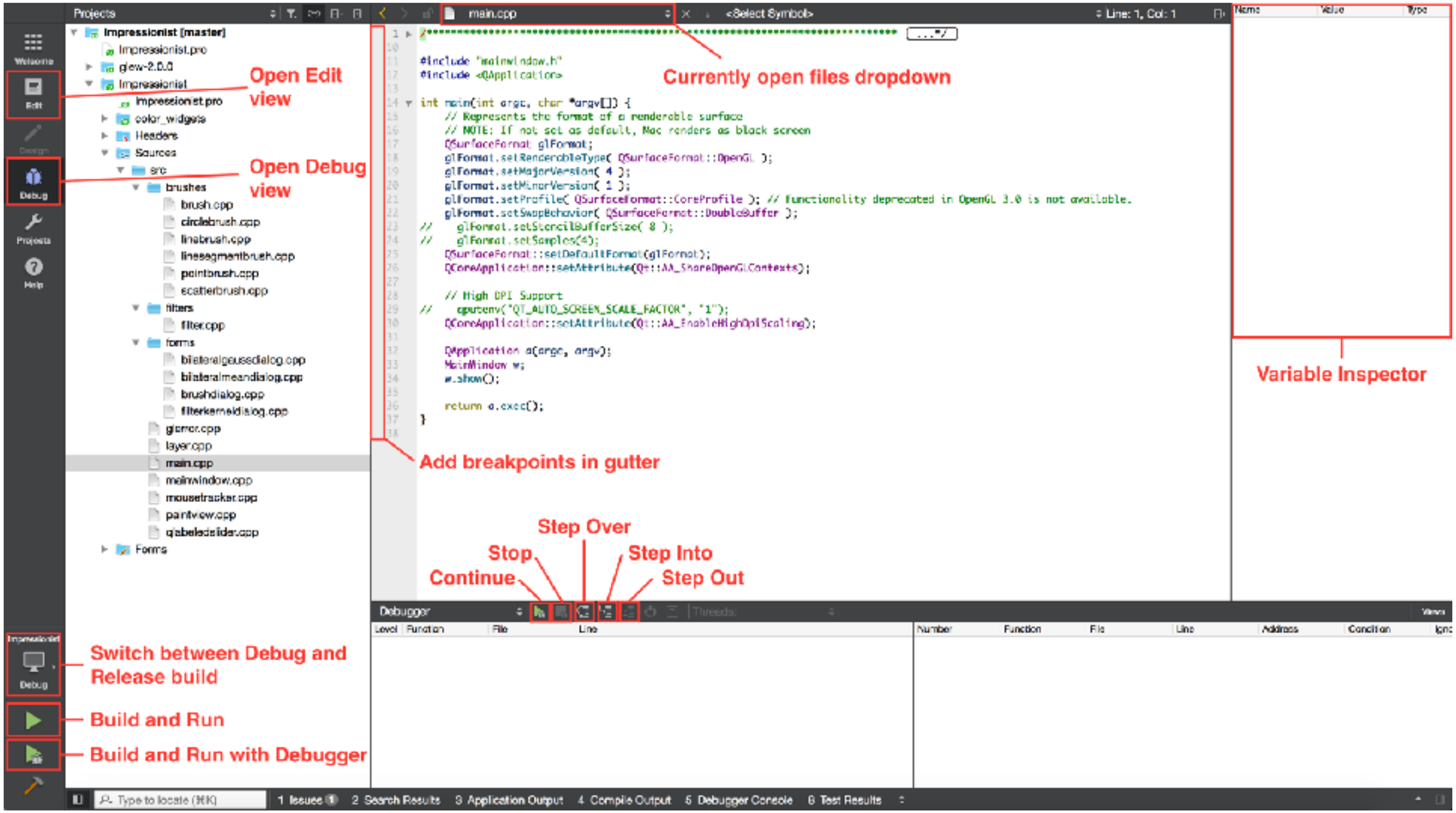
DRAWING A POLYGON

- A lot going on behind the scenes
- There is a lot of prep code needed to draw
  - We need to create a 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`**, … and 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)
- QtCreator IDE - installed with Qt
- mainwindow.cpp has several widget examples

QT



```
/************************************************************************** ...*/

#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[]) {
    // Represents the format of a renderable surface
    // NOTE: If not set as default, Mac renders as black screen
    QSurfaceFormat glFormat;
    glFormat.setRenderableType( QSurfaceFormat::OpenGL );
    glFormat.setMajorVersion( 4 );
    glFormat.setMinorVersion( 1 );
    glFormat.setProfile( QSurfaceFormat::CoreProfile ); // Functionality deprecated in OpenGL 3.0 is not available.
    glFormat.setSwapBehavior( QSurfaceFormat::DoubleBuffer );
//    glFormat.setStencilBufferSize( 8 );
//    glFormat.setSamples(4);
    QSurfaceFormat::setDefaultFormat(glFormat);
    QCoreApplication::setAttribute(Qt::AA_ShareOpenGLContexts);

    // High DPI Support
//    qputenv("QT_AUTO_SCREEN_SCALE_FACTOR", "1");
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);

    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

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();`

REQUIREMENTS

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*

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

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

- Use the sobel operator

$$\tilde{s}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\tilde{s}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

ALPHA BLENDING

- A weighted average of two colors: $F_{new} = \alpha C + (1-\alpha)F_{old}$
- Suppose $\alpha = 0.5 \quad C = \begin{bmatrix}255\\255\\255\\255\end{bmatrix} \qquad F_{old} = \begin{bmatrix}255\\0\\0\\128\end{bmatrix}$
- Then

$$F_{new} = \ ?$$

ALPHA BLENDING

- A weighted average of two colors: $F_{new} = \alpha C + (1 - \alpha) F_{old}$

- Suppose $\alpha = 0.5 \quad C = \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \end{bmatrix} \quad F_{old} = \begin{bmatrix} 255 \\ 0 \\ 0 \\ 128 \end{bmatrix}$

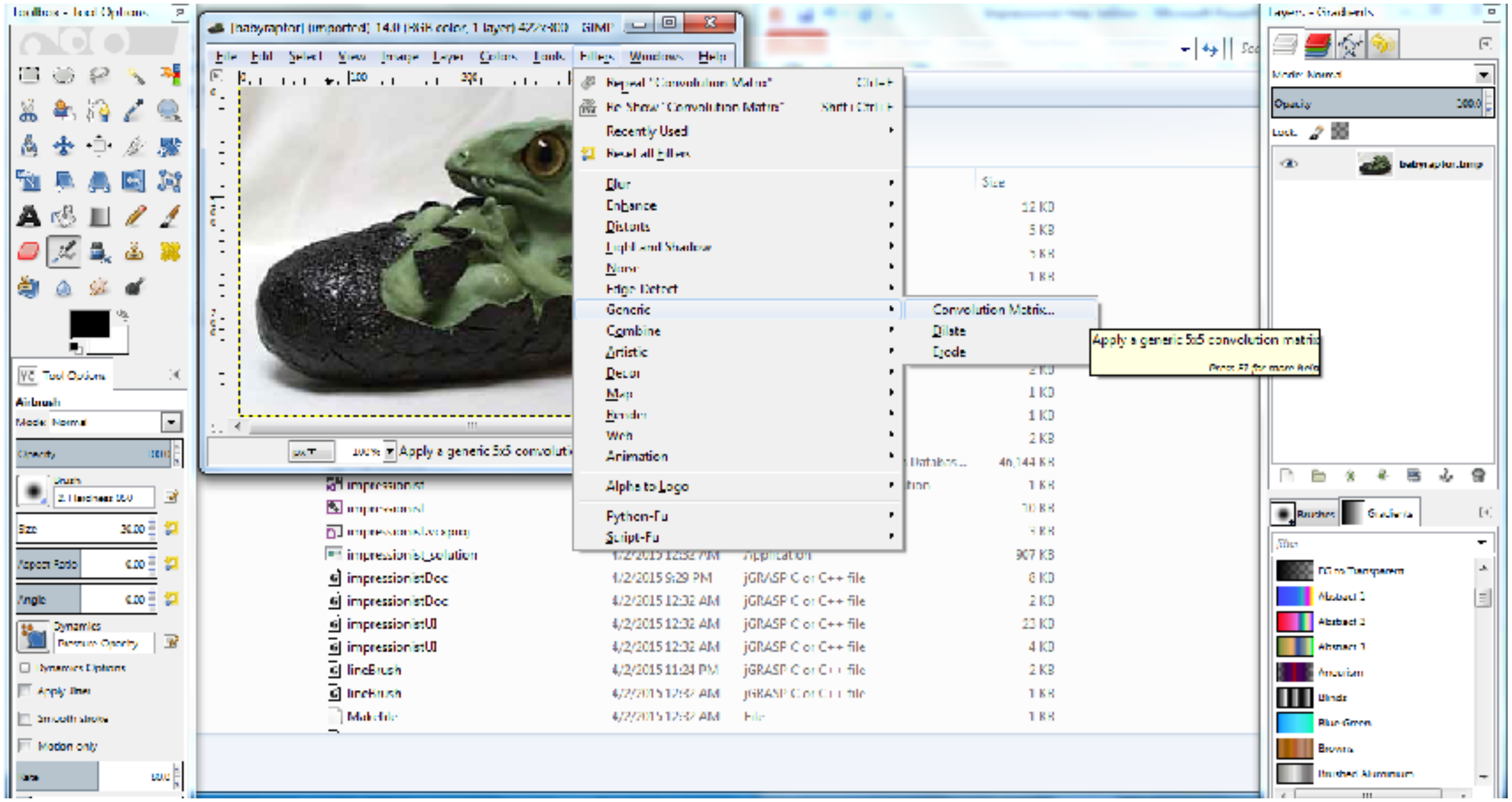
- Then

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

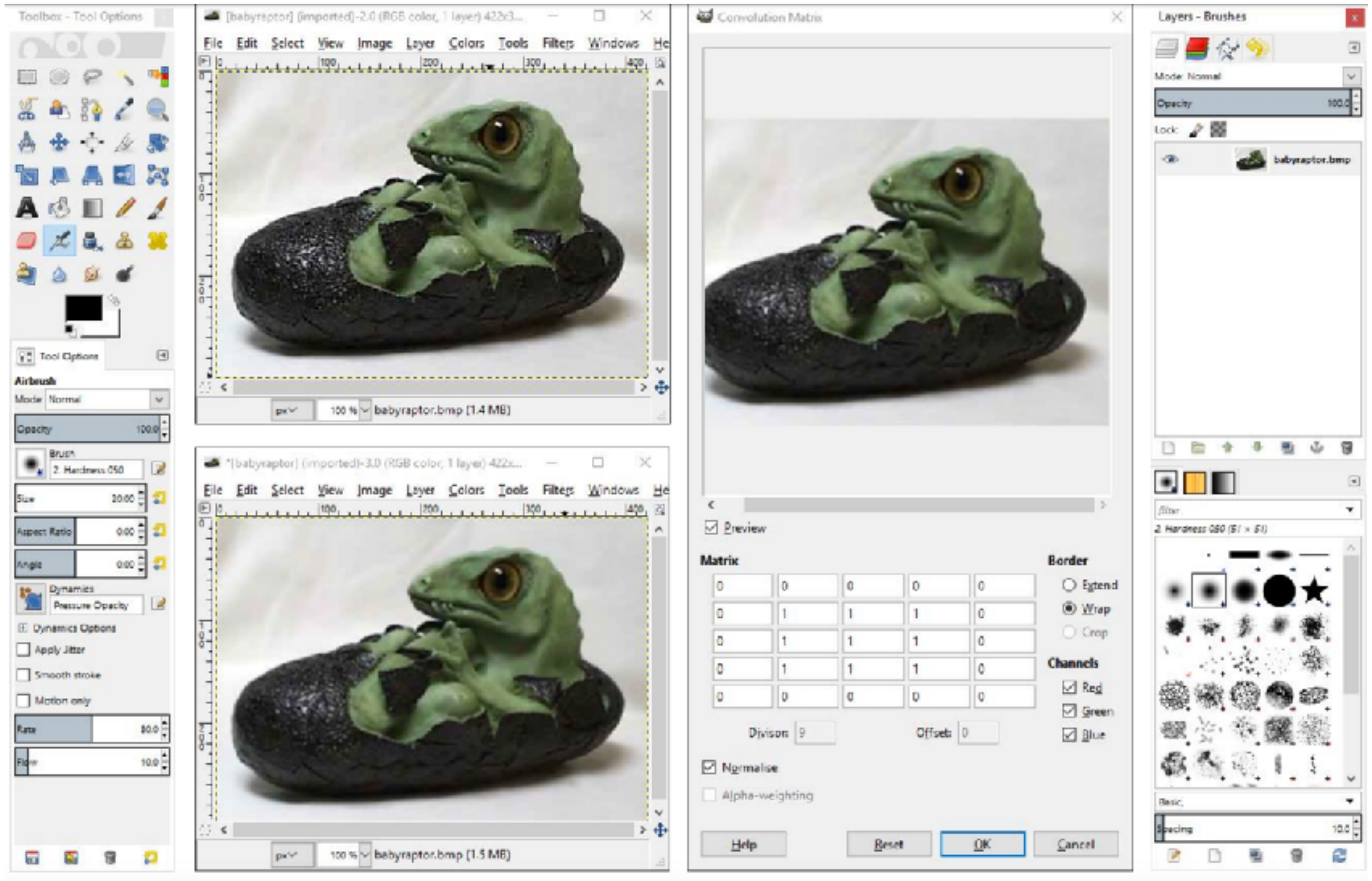
FILTERS

- 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


Toolbox - Tool Options
[babyraptor] (imported)-2.0 (RGB color, 1 layer) 422x3...
File Edit Select View Image Layer Colors Tools Filters Windows He
babyraptor.bmp (1.4 MB)
*[babyraptor] (imported)-3.0 (RGB color, 1 layer) 422x...
babyraptor.bmp (1.5 MB)
Convolution Matrix
Layers - Brushes
Help
Reset
OK
Cancel

Convolution Matrix

[✓] Preview

Matrix

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Divisor 9 Offset 0

Border: Extend [✓ Wrap] Crop

Channels: [✓] Red [✓] Green [✓] Blue

[✓] Normalise

[ ] Alpha-weighting

ARTIFACTS

EVERY PROJECT HAS AN ARTIFACT

- Individual (except for final project)
- Due after the project
- Showcase the tool you built
  - A good place to demonstrate any bells and whistles you implemented
- In-class voting to determine the best
  - Winner gets extra credit!

GIT TUTORIAL

RESOURCES

- Basics for this course:
  - https://courses.cs.washington.edu/courses/csep557/19sp/src/help.php
- Official documentation:
  - https://git-scm.com/book/en/v2
  - `git —help <command>`

WORKFLOW

- Starting
  - Navigate to the directory you want to work in and run
    ```
    $ git clone git@gitlab.cs.washington.edu:csep557-19sp-
    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
    ```

SUBMITTING

- Build your executable in **Release Mode** and test it
- 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

THE END

GOOD LUCK