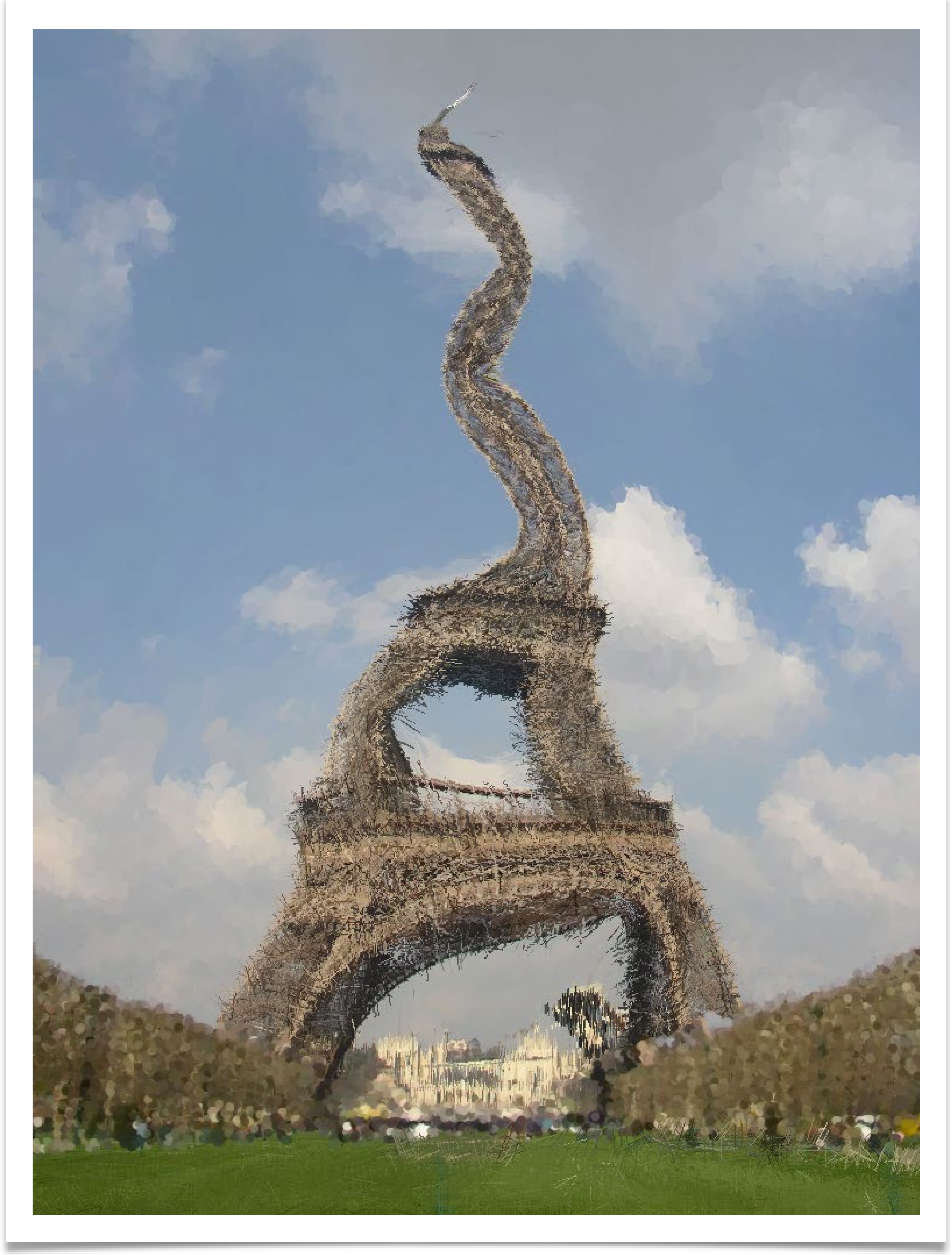
IMPRESSIONIST

HELP SESSION

OUTLINE

- Project Requirements
- Environment Setup
- Skeleton Code
- Some Guides
- Artifacts
- Git Tutorial



PROJECT REQUIREMENTS

- 5 different brush types
  - Single Line, Scattered Lines, Scattered Points, Filled Circle, and Scattered Circles
- Sliders controlling brush attributes
- 4 ways to control brush direction
  - Slider, right mouse button drag, cursor movement, gradient of the image
- Opacity of brush stroke
- Filter kernel
- Mean bilateral filter
- At least one Bell’s worth of extra credit
  - 1 Bell = 2 Whistles

ENVIRONMENT SETUP (INSTRUCTION)

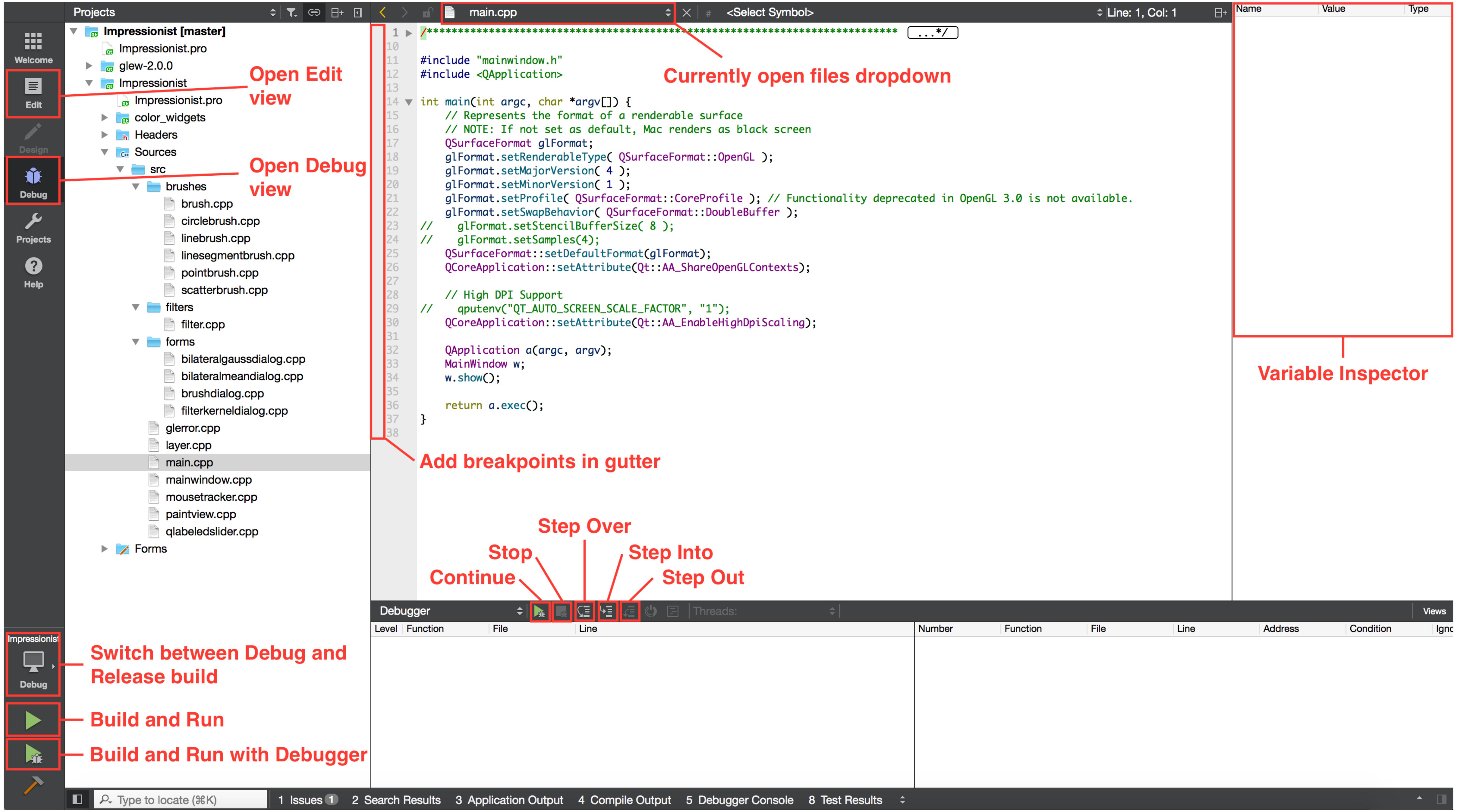
- Install Compiler
  - On Windows, Install Visual Studio Community. Select “Desktop development with C++”.
  - On Max, Install Xcode and use Clang compiler
  - On Linux, use g++
- Download and Install Open-Source Qt www.qt.io/download
  - Select Qt version 5.15.2 or above
  - Select an option related to your compiler

GETTING STARTED

- Clone the Impressionist skeleton code
  - `git clone git@gitlab.cs.washington.edu:cse457-17au-impressionist/YOUR_REPO.git impressionist`
- In Qt Creator, “Open Existing Project” and open Impressionist.pro

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)
- Qt Creator IDE - installed with Qt
- `mainwindow.cpp` has several widget examples



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

#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();
}
```

Open Edit view

Open Debug view

Currently open files dropdown

Variable Inspector

Add breakpoints in gutter

Step Over

Stop

Step Into

Continue

Step Out

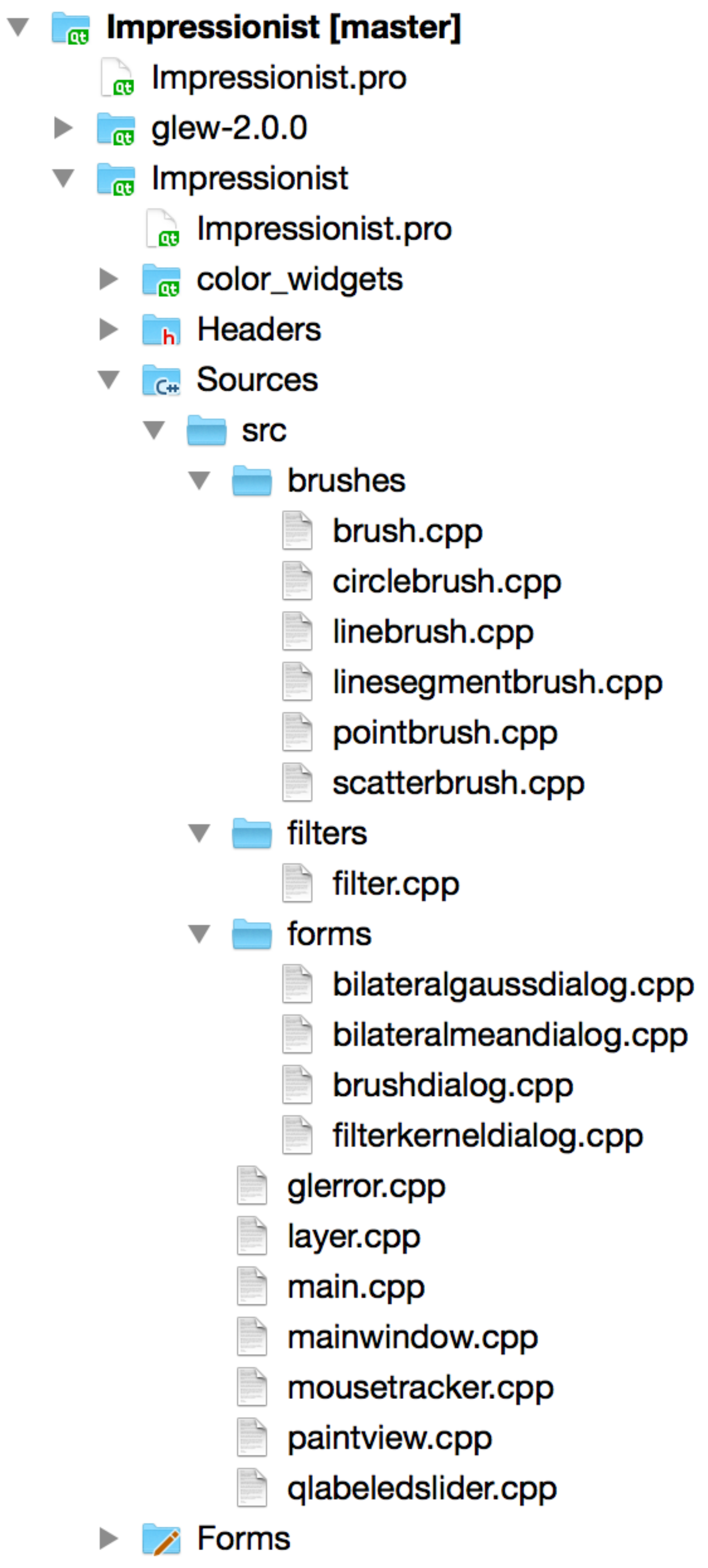
Switch between Debug and Release build

Build and Run

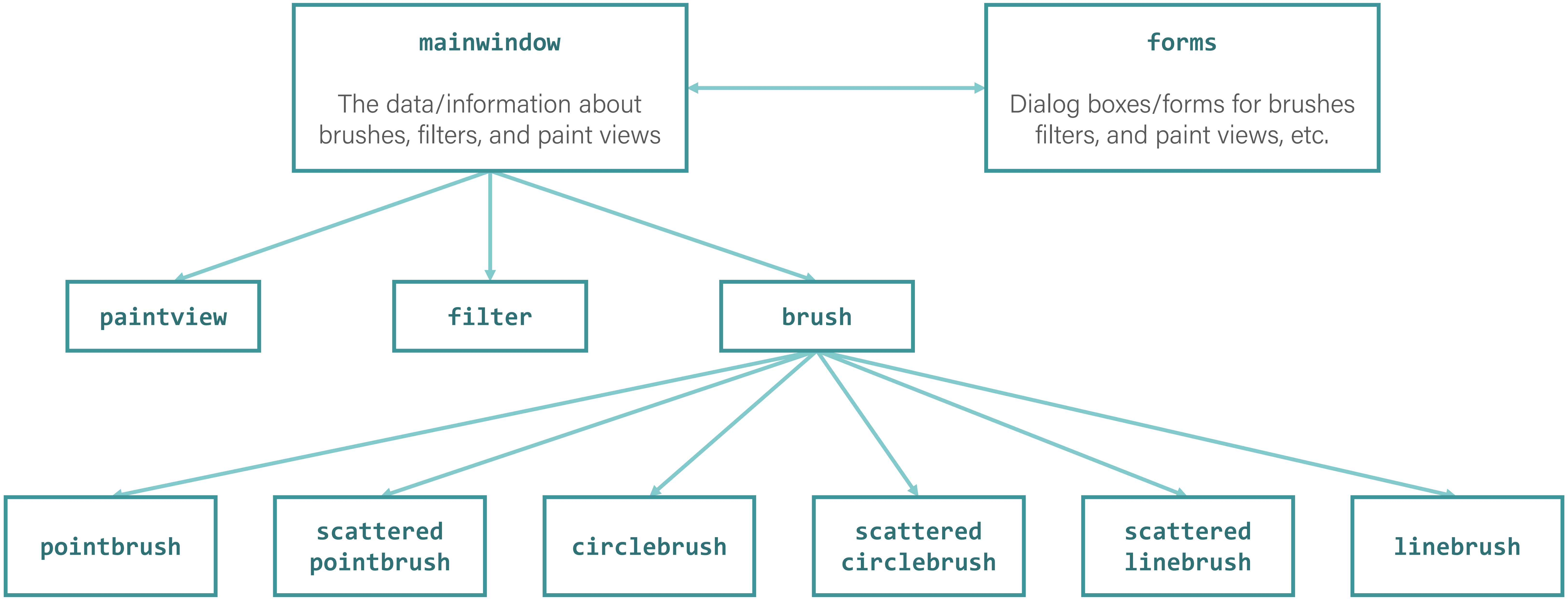
Build and Run with Debugger

SKELETON CODE

- `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 points



```
▼ 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
```



mainwindow

The data/information about brushes, filters, and paint views

forms

Dialog boxes/forms for brushes filters, and paint views, etc.

paintview

filter

brush

pointbrush

scattered pointbrush

circlebrush

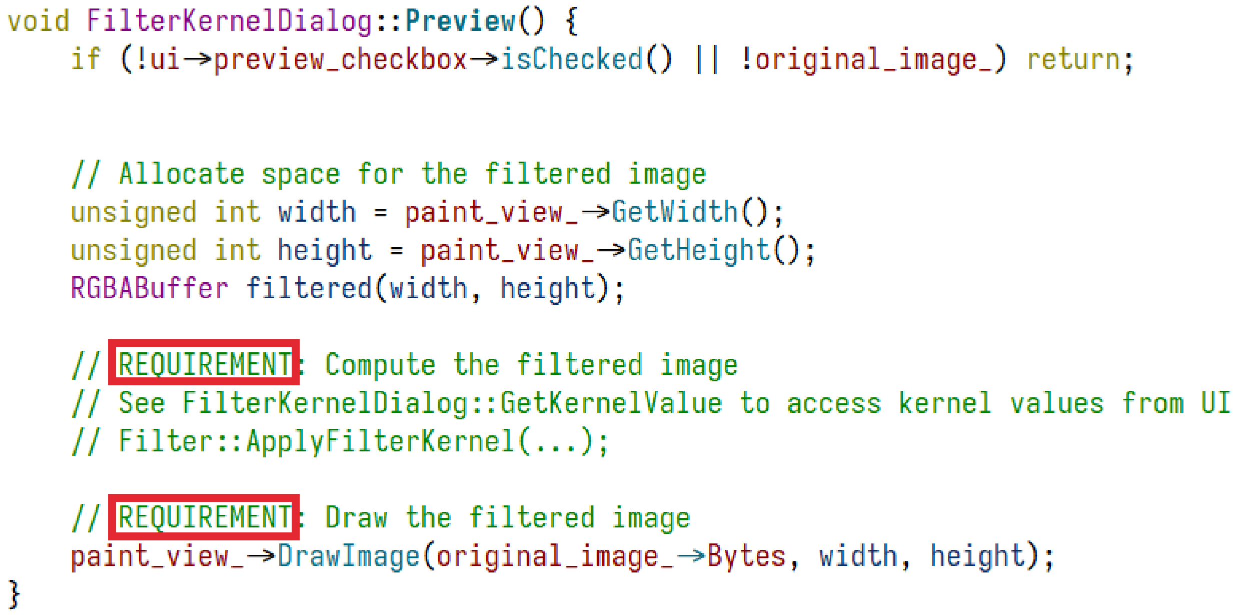
scattered circlebrush

scattered linebrush

linebrush

PROJECT REQUIREMENTS

- You can search for "`REQUIREMENT`" to see which part of the code needs to be changed.
- For example,



```
void FilterKernelDialog::Preview() {
    if (!ui->preview_checkbox->isChecked() || !original_image_) return;


    // Allocate space for the filtered image
    unsigned int width = paint_view_->GetWidth();
    unsigned int height = paint_view_->GetHeight();
    RGBABuffer filtered(width, height);

    // REQUIREMENT: Compute the filtered image
    // See FilterKernelDialog::GetKernelValue to access kernel values from UI
    // Filter::ApplyFilterKernel(...);

    // REQUIREMENT: Draw the filtered image
    paint_view_->DrawImage(original_image_->Bytes, width, height);
}
```

- You may change any part of the code as you see fit.

SOME GUIDES

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;
// [OpenGL call to set color]

// 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.
  - We need to specify how we want to draw these vertices
    - (`GL_LINES`, `GL_TRIANGLES`, `GL_QUADS`, … and many more!)
  - We need to create a shader program (we did this for you)

CREATING NEW 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

CREATING NEW BRUSHES

- 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);
```

CREATING NEW BRUSHES

- 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.

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();`

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

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

EDGE DETECTION & GRADIENTS

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

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

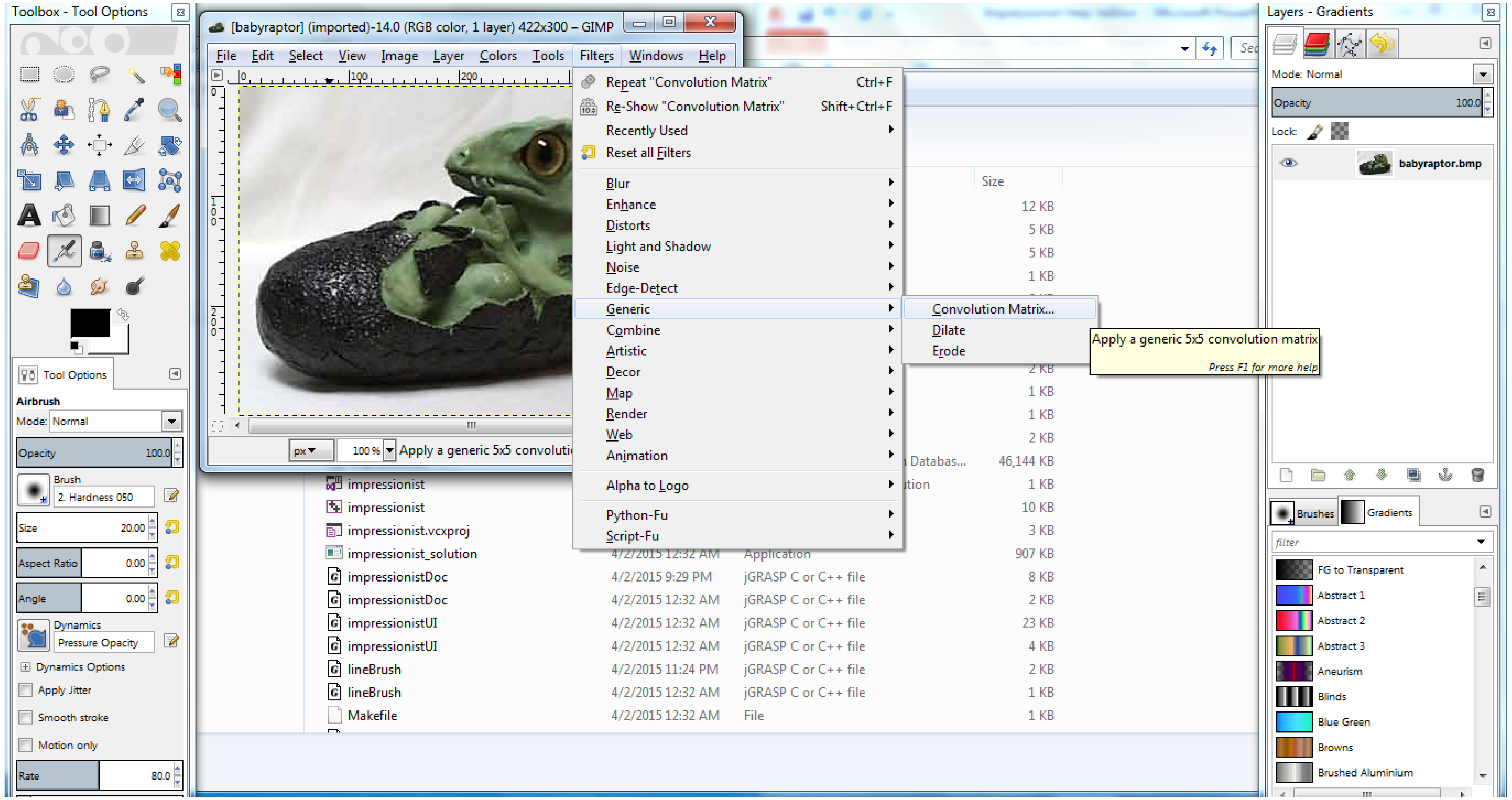
$$\theta = \mathrm{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)$$

- Use the Sobel operator

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



[babyraptor] (imported)-14.0 (RGB color, 1 layer) 422x300 – GIMP

Apply a generic 5x5 convolution matrix

Press F1 for more help

Airbrush

Mode: Normal

Opacity 100.0

Brush 2. Hardness 050

Size 20.00

Aspect Ratio 0.00

Angle 0.00

Dynamics Pressure Opacity

□ Apply Jitter

□ Smooth stroke

□ Motion only

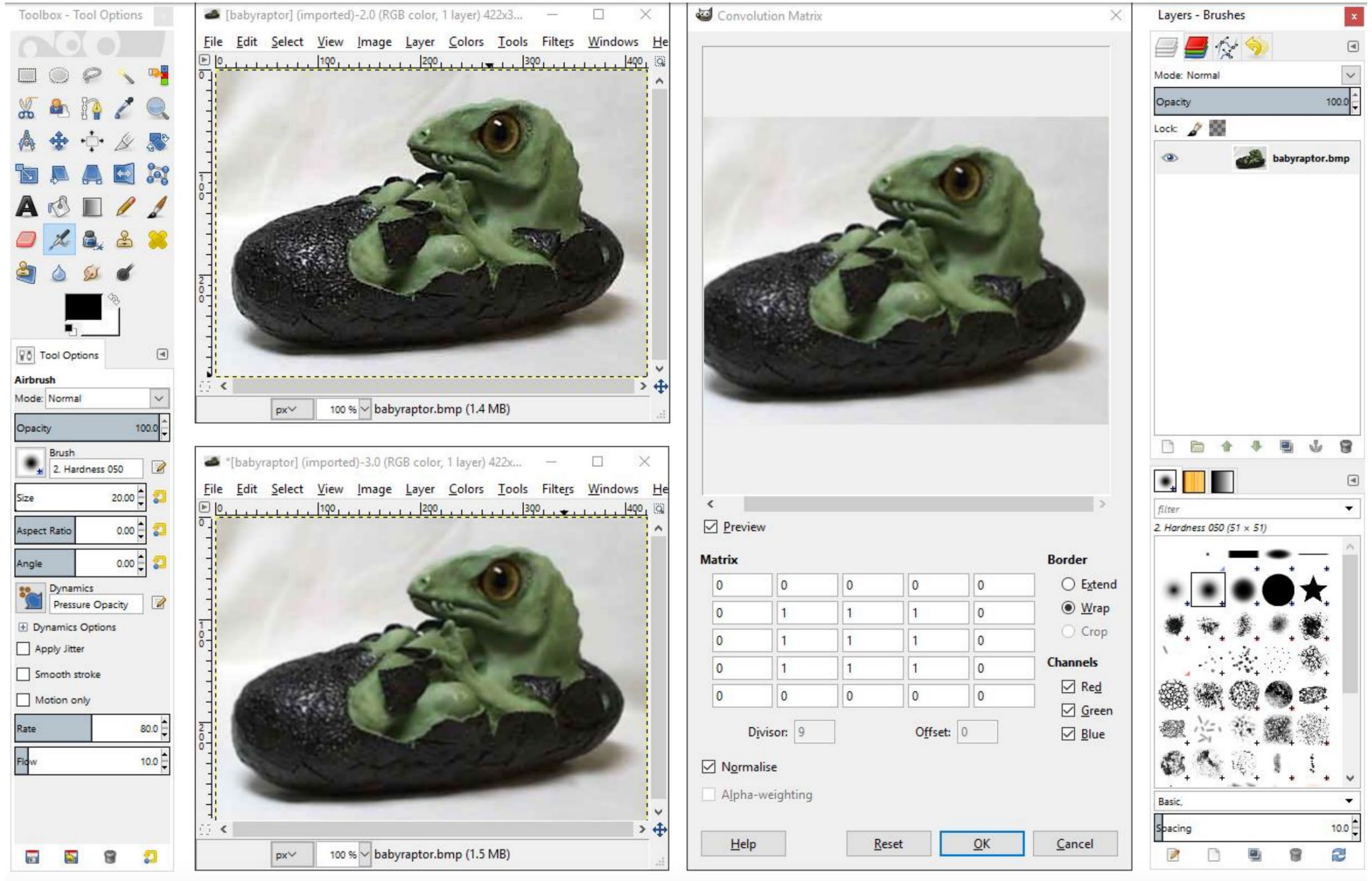
Rate 80.0

Mode: Normal

Opacity 100.0

			Size
			12 KB
			5 KB
			5 KB
			1 KB
			2 KB
			1 KB
			1 KB
			2 KB
		Databas...	46,144 KB
impressionist		tion	1 KB
impressionist			10 KB
impressionist.vcxproj			3 KB
impressionist_solution	4/2/2015 12:32 AM	Application	907 KB
impressionistDoc	4/2/2015 9:29 PM	jGRASP C or C++ file	8 KB
impressionistDoc	4/2/2015 12:32 AM	jGRASP C or C++ file	2 KB
impressionistUI	4/2/2015 12:32 AM	jGRASP C or C++ file	23 KB
impressionistUI	4/2/2015 12:32 AM	jGRASP C or C++ file	4 KB
lineBrush	4/2/2015 11:24 PM	jGRASP C or C++ file	2 KB
lineBrush	4/2/2015 12:32 AM	jGRASP C or C++ file	1 KB
Makefile	4/2/2015 12:32 AM	File	1 KB

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
px 100 % babyraptor.bmp (1.4 MB)
*[babyraptor] (imported)-3.0 (RGB color, 1 layer) 422x...
px 100 % babyraptor.bmp (1.5 MB)
Convolution Matrix
Layers - Brushes
babyraptor.bmp
2. Hardness 050 (51 × 51)
Basic,
Help
Reset
OK
Cancel

Tool Options

Airbrush

Mode: Normal

Opacity 100.0

Brush 2. Hardness 050

Size 20.00

Aspect Ratio 0.00

Angle 0.00

Dynamics Pressure Opacity

Dynamics Options

□ Apply Jitter

□ Smooth stroke

□ Motion only

Rate 80.0

Flow 10.0

Mode: Normal

Opacity 100.0

Lock:

Spacing 10.0

☑ 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/cse457/21wi/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:cse457-17au-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 int a SEPARATE directory and test running the program

BRANCHES AND MERGE REQUESTS (ADVANCED, OPTIONAL)

- You can create your own branch and work separately.
- To create a new branch
  - `$ git checkout –b my-new-branch`
- To switch back to the `master` branch
  - `$ git checkout master`
- Once you are done with your branch, you can create a merge request on Gitlab website.
  - You can review changes.
  - You will need to resolve any conflicts.
  - Then, you can merge to the `master` branch.
- If you choose to do this, make sure your codes are merged on the `master` branch, and you are submitting on the `master` branch.