IMPRESSIONIST

OUTLINE

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

- Clone the Impressionist skeleton code
  - `git clone
git@gitlab.cs.washington.edu:cse457-20sp-impressionist/YOUR_REPO.git impressionist`

- Install Qt Creator
  - [www.qt.io/download](http://www.qt.io/download) > Downloads for open source users
  - 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
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
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();`
The data/information about brushes, filters, and paint views

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

- paintview
- filter
- brush
- pointBrush
- scatteredPointBrush
- circleBrush
- scatteredCircleBrush
- scatteredLineBrush
- lineBrush
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 points
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
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.
Modify the `BrushMove` method to draw a triangle instead of a point in `trianglebrush.cpp`

```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

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

- Suppose  \( \alpha = 0.5 \)  \( C = \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \end{bmatrix} \)

- Then  \( F_{old} = \begin{bmatrix} 255 \\ 0 \\ 0 \\ 128 \end{bmatrix} \)

\( F_{new} = \ ? \)
ALPHA BLENDING

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

- Suppose \( \alpha = 0.5 \)

- 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
REQUIREMENTS

USE GIMP/PHOTOSHOP TO SEE FILTERS IN ACTION
3x3 MEAN BOX FILTER
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!
RESOURCES

▸ Basics for this course:

▸ Official documentation:
  ➤ 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 into a SEPARATE directory and test running the program
THE END

GOOD LUCK