PP3 Report

DFT

`dft.hpp, dft.cpp` implements the basic and separable dft functions.

1d DFTs

`dft.hpp` has a forward and bak version of the basic 1d dft. `ft_tests()` in `app.cpp` has calls to these functions. I checked them against opencv calls and against numpy output to convince myself they are right.

2d DFTs

`dft.hpp` has a forward and bak version of the 2d dft. There is a slow version, and a separability version, which are tested in `app.cpp`, in `ft_tests()`

FFT

`fft.hpp, fft.cpp` implements fft.

`fft.hpp` has a forward and inverse Cooney-Tukey FFT. I used Numerical Recipes as reference, but made modifications to take a std::vector of std::complex as input, along with further changes to use a W twiddle lookup rather than computing W progressively.

four1

I referred to Numerical Recipes, which gives the four1 function. This is a function for performing fft and ifft from Numerical Recipes, which credits N. M. Brenner (from the 70s).

Inverse transform using forward fft

To get the inverse transform, we get the complex conjugates of all the pixels, and do a forward fft on that. Then we take a complex conjugate of the result.

But if the input is real, the conjugate on the final fft output would have no effect one way or the other, because the imaginary terms would have to be near 0 either way.. right?

four1 takes a `isign` parameter, which gets multiplied into the exponent of W twiddle factors. This controls the sign of the imaginary term in the result. This is a part of Progressive Twiddle Factor update, where `isign` is multiplied into $\Delta\theta$ (there's a section on Progressive Twiddle Factor Update below).

In this project, I ended up writing fft within a `twiddle_lookup` class, where twiddle factors are pre-computed. For the `ifft` I just negated the imaginary terms of the input, and took the real terms of the output after putting the conjugate F through forward fft. Eventually, I got this to work.

Setting `isign=-1` also worked pretty well when using the original `four1`. However, manually getting the conjugate of F and then running it through forward `four1` resulted in a noisy image.. floating point errors?

Progressive Twiddle Factor Update

This is a trick that `four1` uses to get the twiddle factors with only 2 calls to `sin()`.

As we iterate through the first half of the samples we want to calculate:

$$W_N^k = e^{-2i\pi k/N}$$

Note that at first, for $k = 0$, $W_n^0 = (1, 0)$
In general:
$\theta = -2\pi k/N$
$W_n^k = cos(\theta) + isin(\theta)$

Note that as we iterate k in $[0, K)$ we are just adding $\Delta\theta$ to θ:
$\Delta\theta = \frac{-2\pi}{N}$

$$\theta_t = \theta_{t-1} + (-2\pi/N) = \theta_{t-1} + \Delta\theta$$

So, say we have W_N^k and we want W_N^{k+1} . We actually want $e^{i(\theta+\Delta\theta)}$ or $e^{i\theta}e^{i\Delta\theta}$

We use trig identities to progressively get W_N^{k+1} without many trig calls:
$cos(a + b) = cos(a)cos(b) - sin(a)sin(b)$
$cos(\theta + \Delta\theta) = cos(\theta)cos(\Delta\theta) - sin(\theta)sin(\Delta\theta)$
$cos(2\theta) = 1 - 2sin^2(\theta)$

$e^{i\Delta\theta} = cos(\Delta\theta) + isin(\Delta\theta)$
$e^{i\theta+\Delta\theta} = e^{i\theta}(cos(\Delta\theta) + isin(\Delta\theta))$
$cos(\Delta\theta) = 1 - 2sin^2(0.5\Delta\theta)$
$e^{i\theta+\Delta\theta} = e^{i\theta}([1 - 2sin^2(0.5\Delta\theta)] + i \cdot sin(\Delta\theta))$
$e^{i\theta+\Delta\theta} = (e^{i\theta})[1 - 2sin^2(0.5\Delta\theta)] + (e^{i\theta})i \cdot sin(\Delta\theta)$
$e^{i\theta+\Delta\theta} = (e^{i\theta}) - (e^{i\theta})2sin^2(0.5\Delta\theta) + (e^{i\theta})i \cdot sin(\Delta\theta)$
$e^{i\theta+\Delta\theta} = e^{i\theta} + e^{i\theta}\Big[-2sin^2(0.5\Delta\theta) + i \cdot sin(\Delta\theta)\Big])$

This is the derivation for the update of the twiddle factor, which in modern c++ looks like:

```
const double theta = -sign * (CV_2PI / fft_N);
const double wtemp = std::sin(0.5 * theta);
std::complex<double> wp{ -2.0 * wtemp * wtemp, std::sin(theta) };
...
w = w + w*wp
```

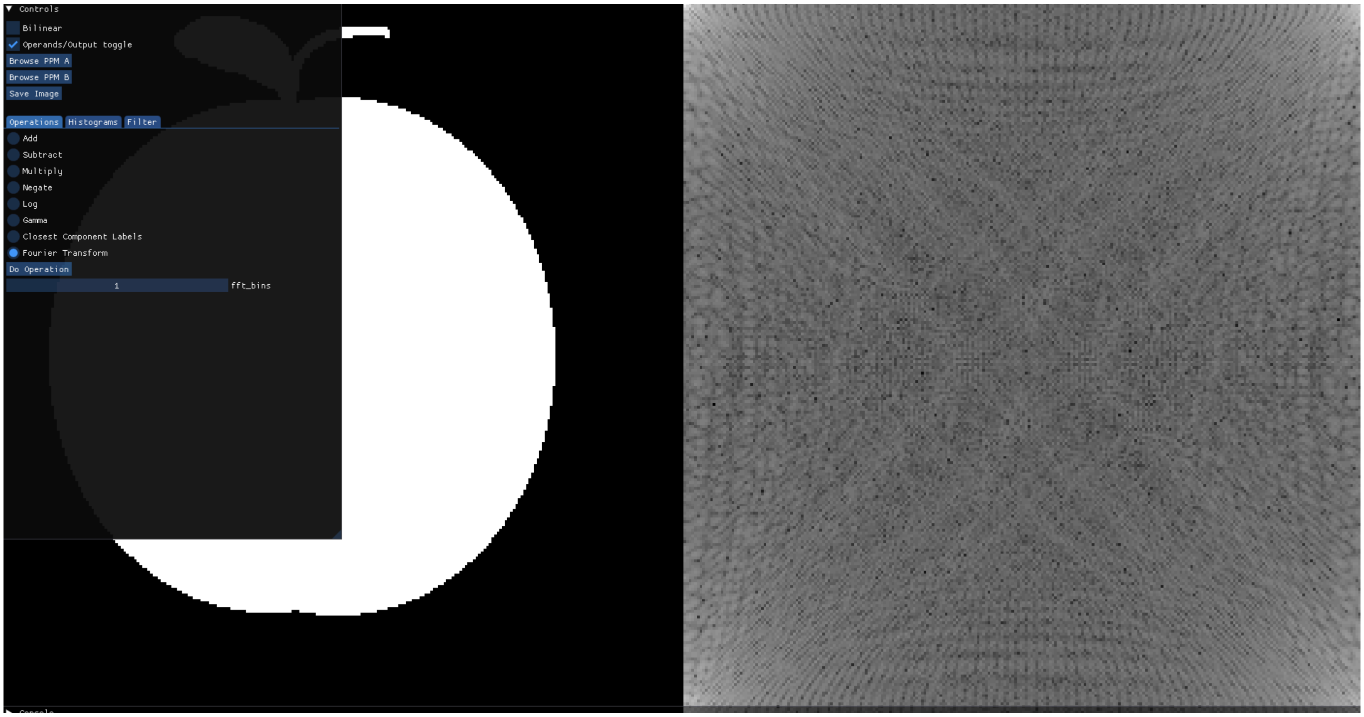
And in old four1 C code looks like

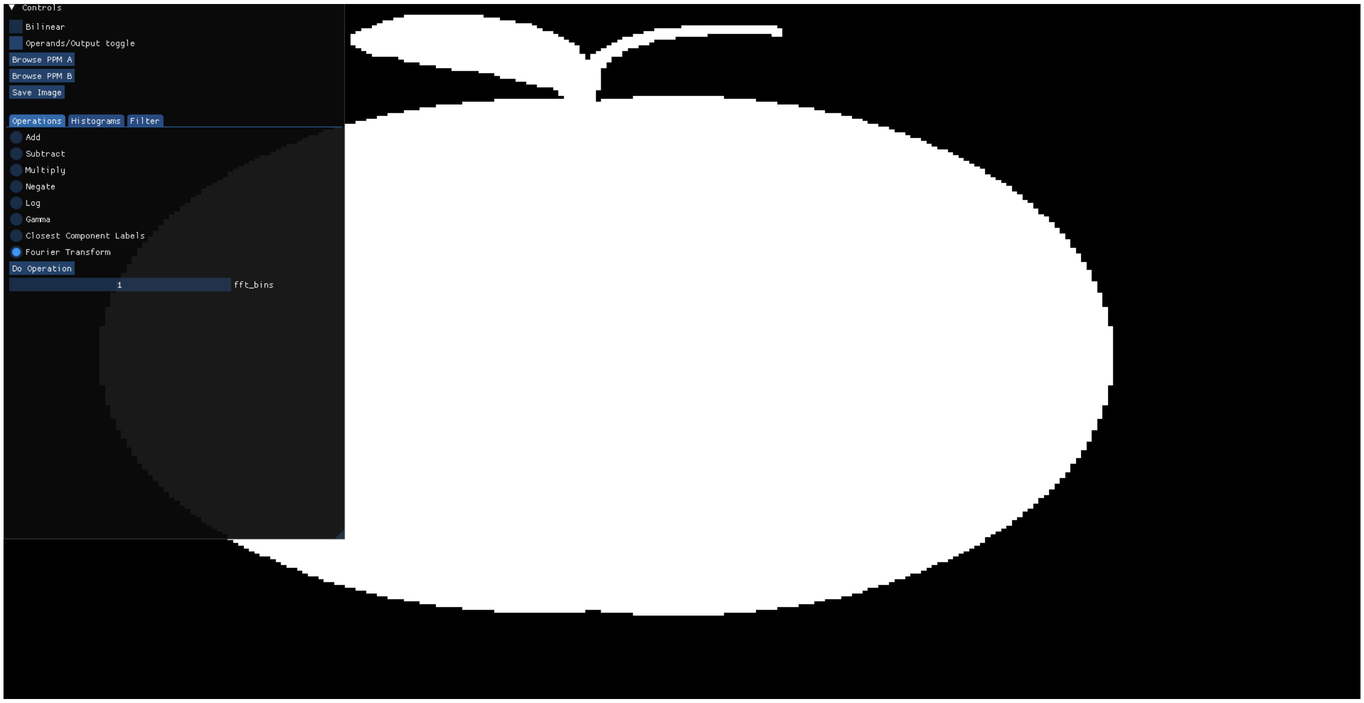
```
istep = mmax << 1; theta=isign*(6.28318530717959/mmax);
wtemp = sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi = sin(theta);
...
wr = (wtemp=wr)*wpr - wi*wpi + wr;
wi = wi*wpr  +wtemp*wpi + wi;
```

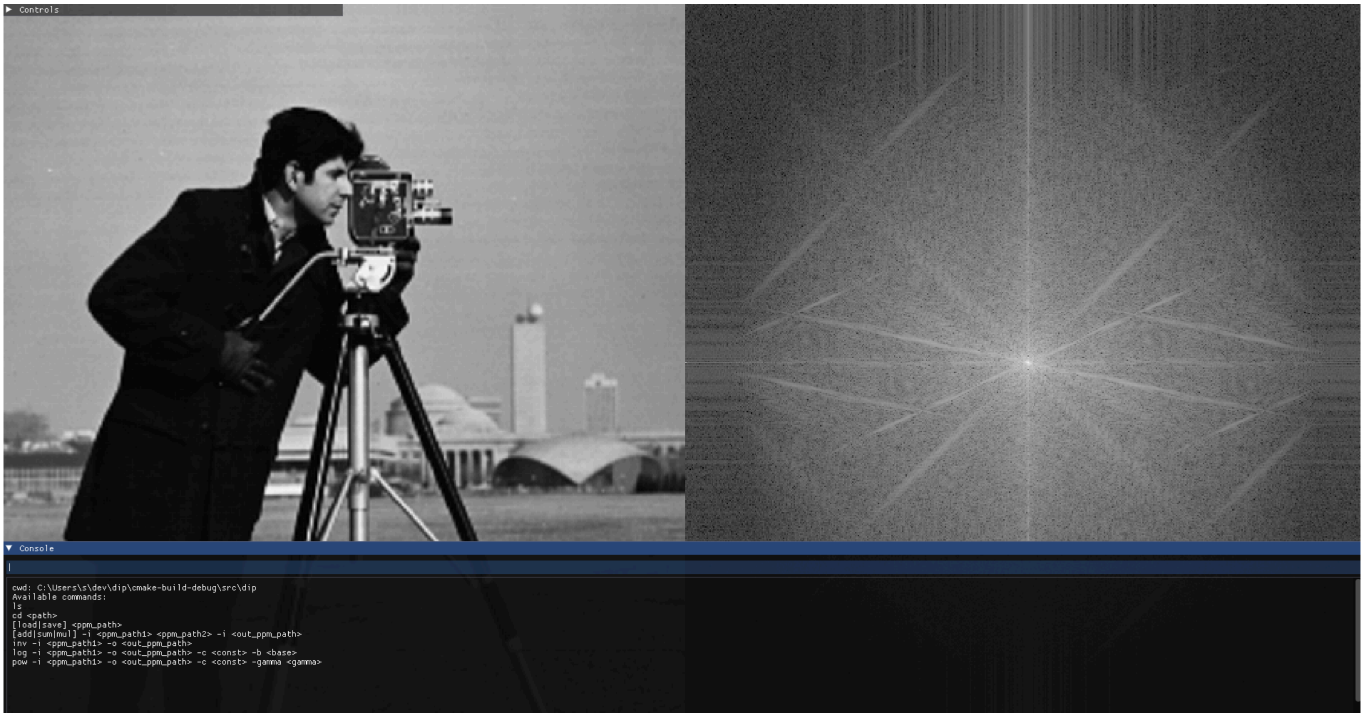
Twiddle Table

When four1 was written (probably the 70s?) we might care about just having a huge lookup table W for any N and k. But we surely don't anymore, and we can just cache W values for any N and k up to some limit. For that, I wrote the `twiddle_table` class.

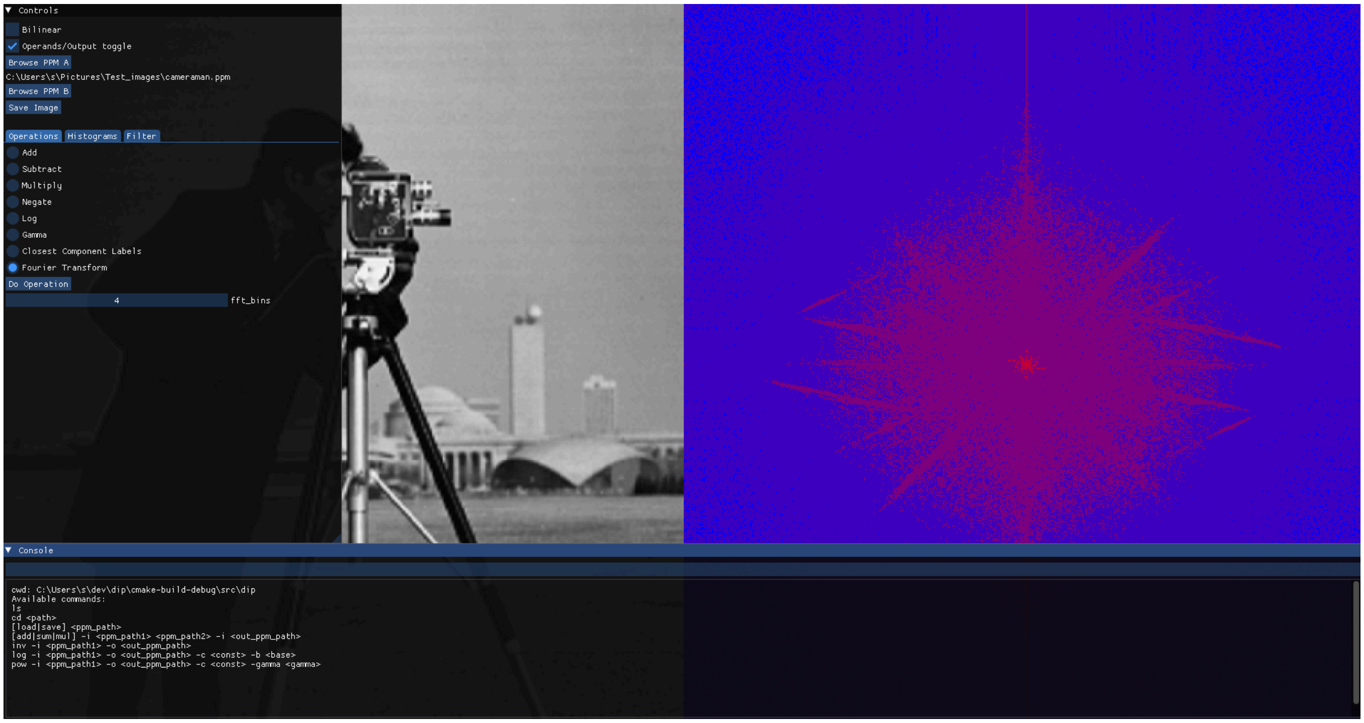
Outputs


▼ Controls
□ Bilinear
☑ Operands/Output toggle
Browse PPM A
Browse PPM B
Save Image
Operations
Histograms
Filter
○ Add
○ Subtract
○ Multiply
○ Negate
○ Log
○ Gamma
○ Closest Component Labels
● Fourier Transform
Do Operation
1
fft_bins
▶ Console


▼ Controls
□ Bilinear
□ Operands/Output toggle
Browse PPM A
Browse PPM B
Save Image
Operations
Histograms
Filter
○ Add
○ Subtract
○ Multiply
○ Negate
○ Log
○ Gamma
○ Closest Component Labels
● Fourier Transform
Do Operation
1
fft_bins



```
|
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```



□ Bilinear

☑ Operands/Output toggle

C:\Users\s\Pictures\Test_images\cameraman.ppm

○ Add

○ Subtract

○ Multiply

○ Negate

○ Log

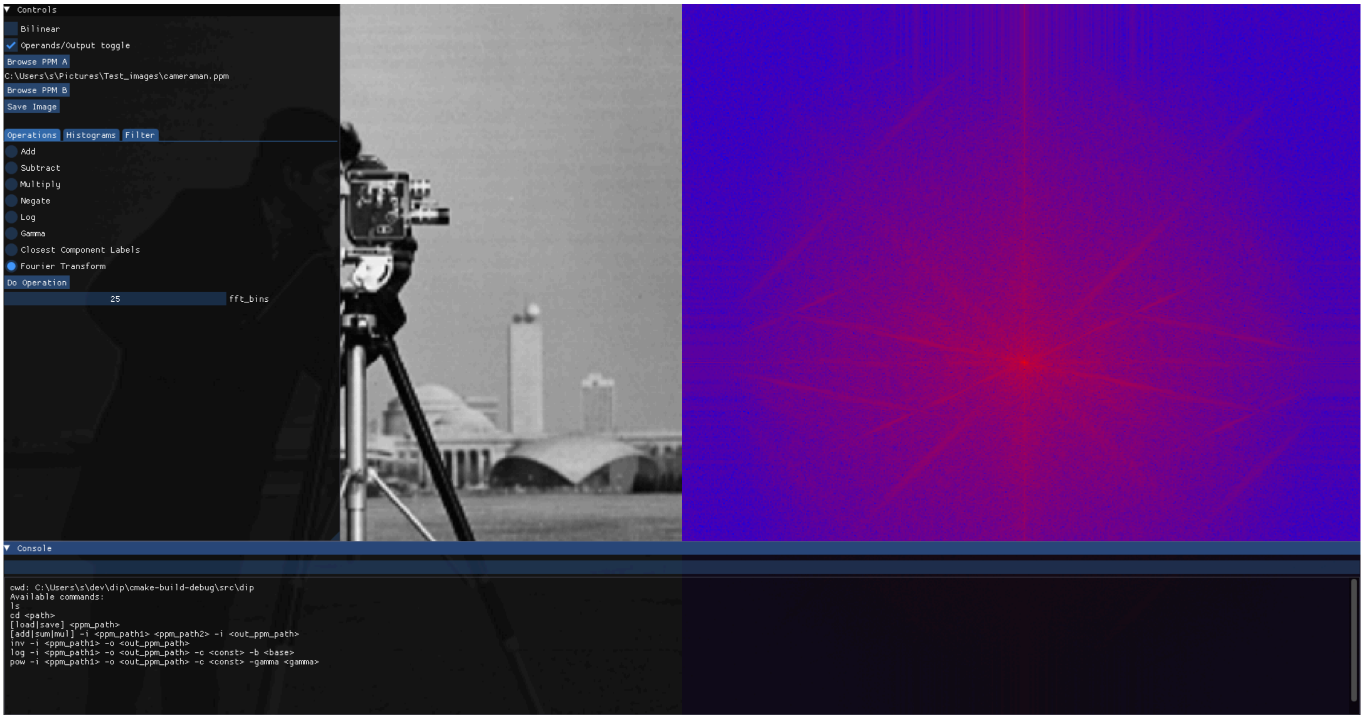
○ Gamma

○ Closest Component Labels

● Fourier Transform

4 fft_bins

```
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```



☐ Bilinear

☑ Operands/Output toggle

C:\Users\s\Pictures\Test_images\cameraman.ppm

○ Add

○ Subtract

○ Multiply

○ Negate

○ Log

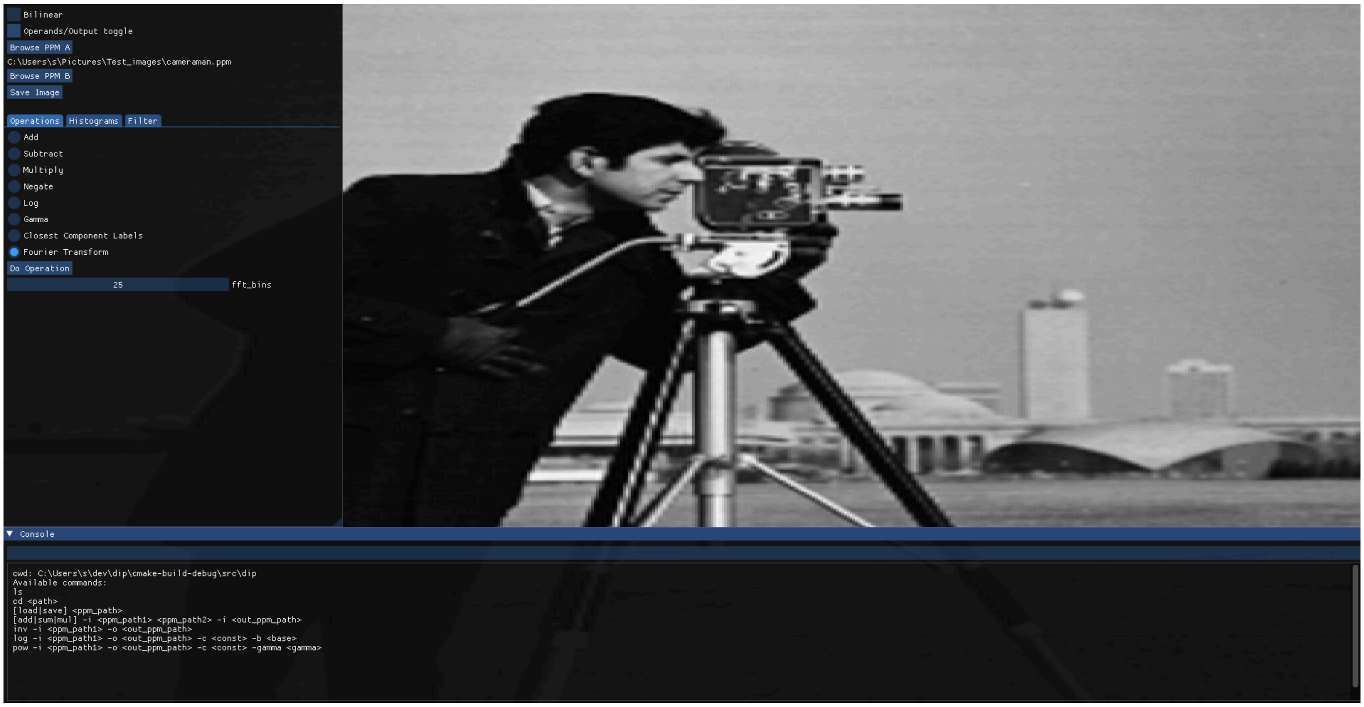
○ Gamma

○ Closest Component Labels

● Fourier Transform

25 fft_bins

```
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```



☐ Bilinear

☐ Operands/Output toggle

C:\Users\s\Pictures\Test_images\cameraman.ppm

○ Add

○ Subtract

○ Multiply

○ Negate

○ Log

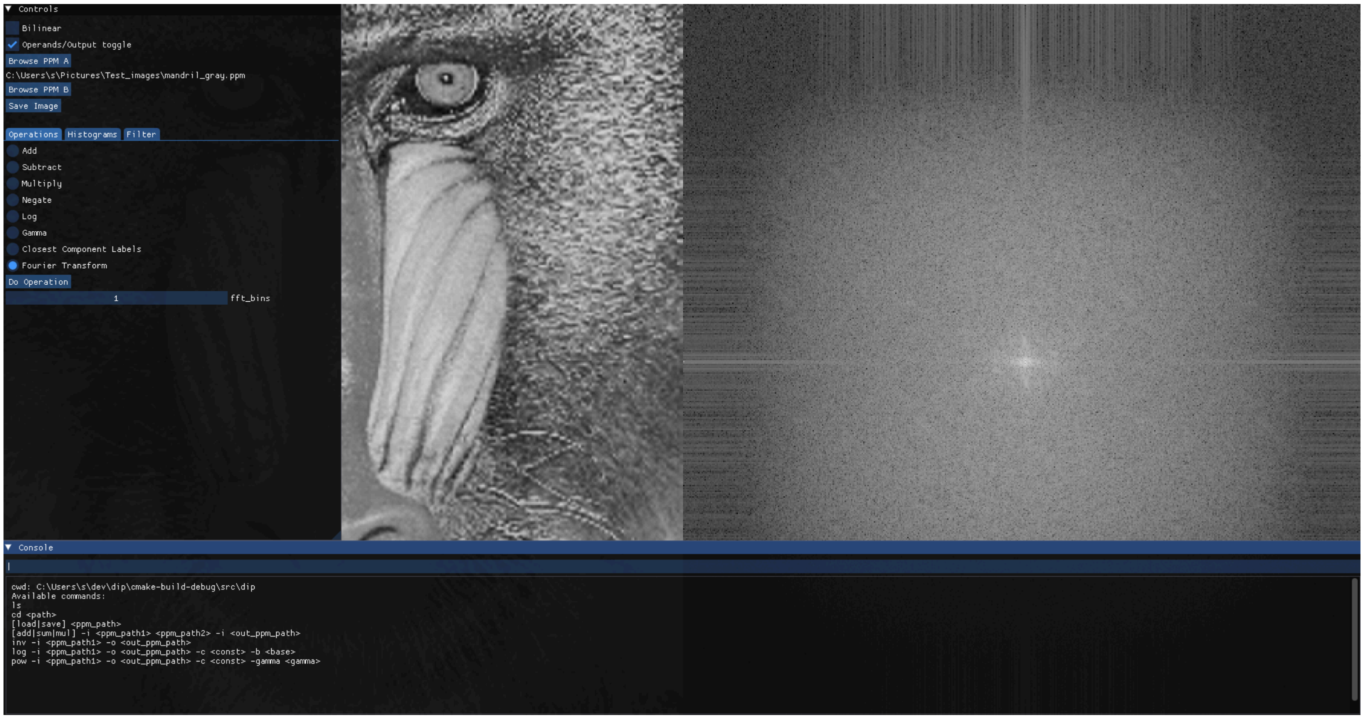
○ Gamma

○ Closest Component Labels

● Fourier Transform

25 fft_bins

```
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```



▼ Controls

□ Bilinear

☑ Operands/Output toggle

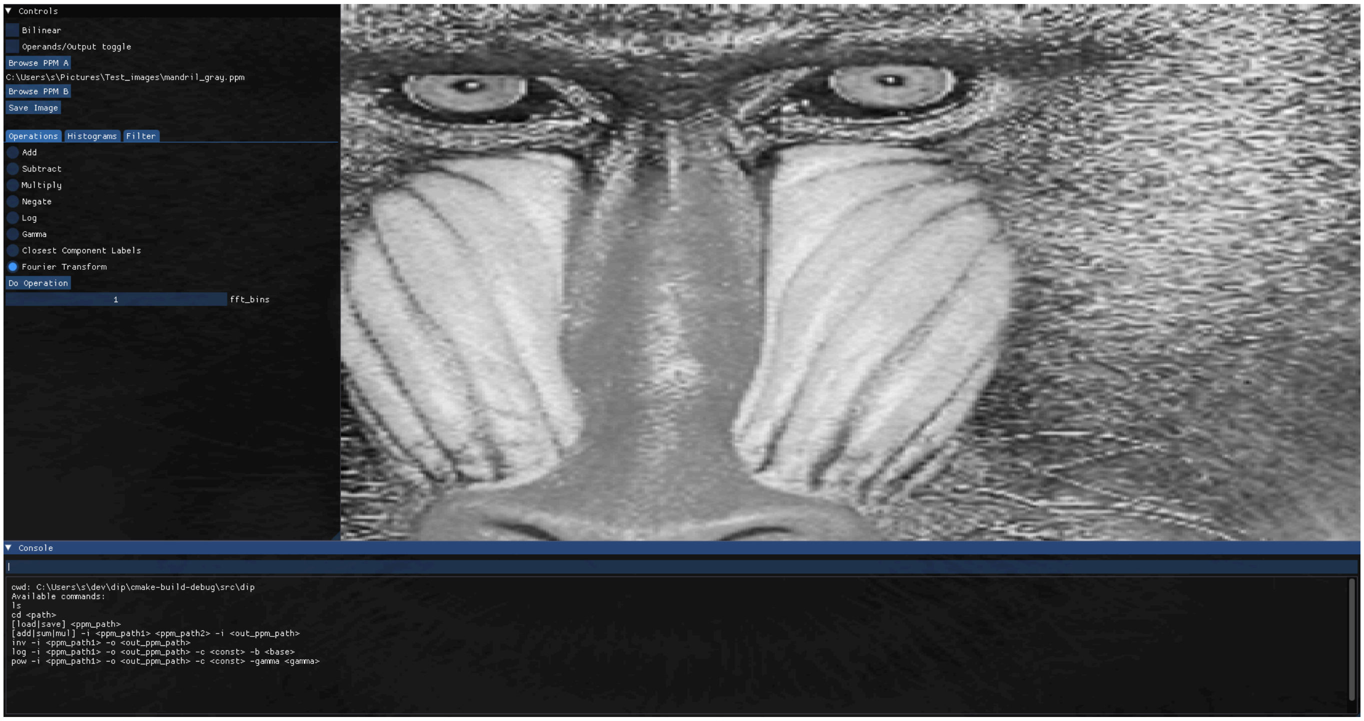
C:\Users\s\Pictures\Test_images\mandril_gray.ppm

○ Add
○ Subtract
○ Multiply
○ Negate
○ Log
○ Gamma
○ Closest Component Labels
● Fourier Transform

1 fft_bins

▼ Console

```
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```



▼ Controls

□ Bilinear

□ Operands/Output toggle

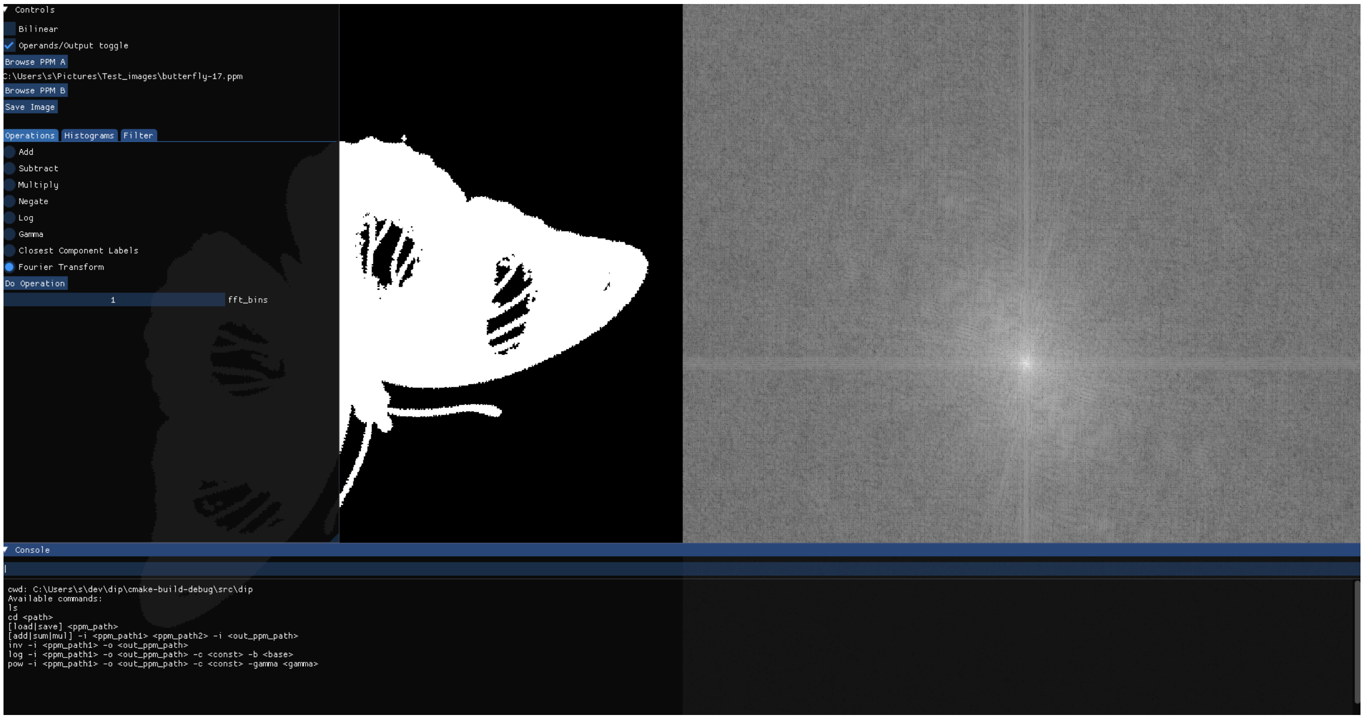
C:\Users\s\Pictures\Test_images\mandril_gray.ppm

○ Add
○ Subtract
○ Multiply
○ Negate
○ Log
○ Gamma
○ Closest Component Labels
● Fourier Transform

1 fft_bins

▼ Console

```
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```



▼ Controls

□ Bilinear

☑ Operands/Output toggle

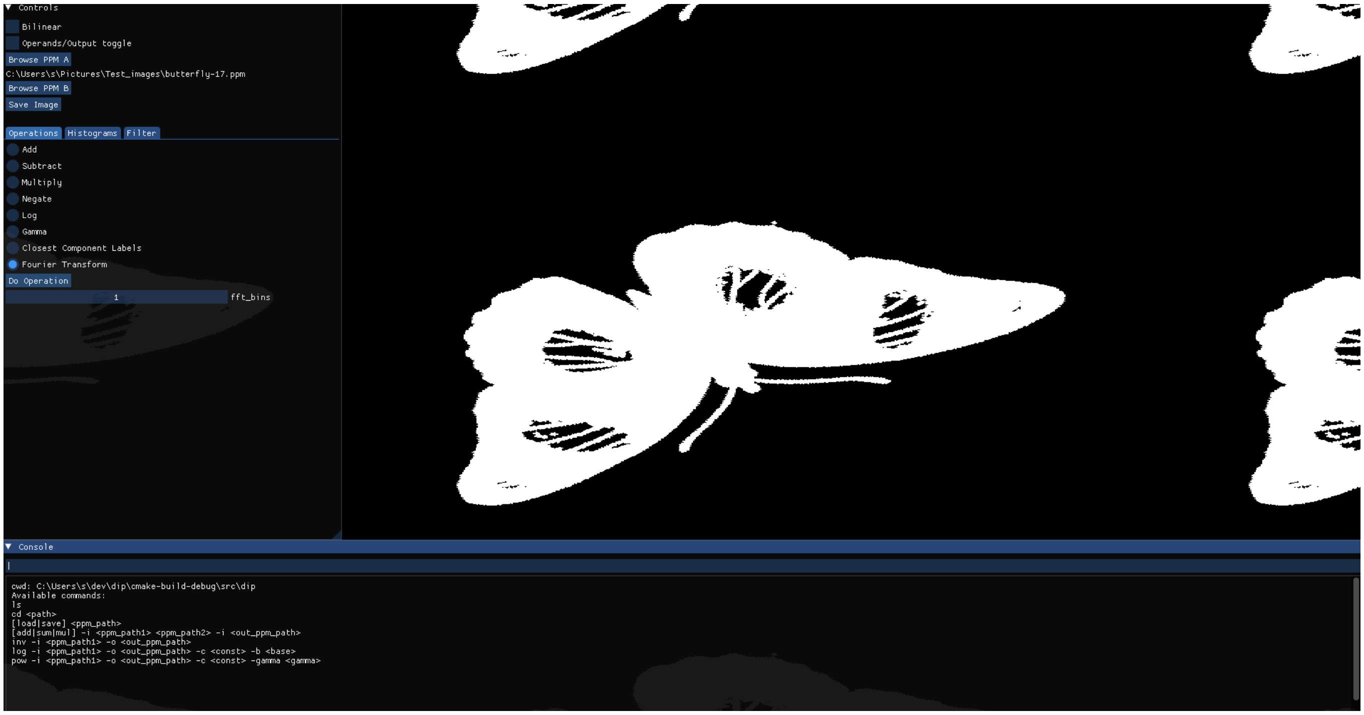
C:\Users\s\Pictures\Test_images\butterfly-17.ppm

○ Add
○ Subtract
○ Multiply
○ Negate
○ Log
○ Gamma
○ Closest Component Labels
● Fourier Transform

fft_bins: 1

▼ Console

```
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```



▼ Controls

□ Bilinear

□ Operands/Output toggle

C:\Users\s\Pictures\Test_images\butterfly-17.ppm

○ Add
○ Subtract
○ Multiply
○ Negate
○ Log
○ Gamma
○ Closest Component Labels
● Fourier Transform

fft_bins: 1

▼ Console

```
cwd: C:\Users\s\dev\dip\cmake-build-debug\src\dip
Available commands:
ls
cd <path>
[load|save] <ppm_path>
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```

Using Frequency Shifting to center the F image

There is a property of Frequency Shifting:
f(x) *exp(-2* pi *i* u_0 * x) <=> F(u - u_0)

We want F(u - N/2)

In 2d, it's exp(-2 *pi* i *(u_0* x + v_0 * y))

u_0 and v_0 are N/2 for a square image where N is a power of 2.
The 0.5 can go outside the parentheses and cancel the -2, so:

exp(-pi *i* (x + y))

So we can multiply each element of the image by this term to shift in the frequency domain:

```
std::complex<double> half_freq = std::polar(1.0, -CV_PI * (row + col));
rows[row][col] = std::complex<double>(padded_f.at<float>(row, col), 0) *
half_freq;
```

It can be further simplified by cos(pi *i) = -1 and sin(pi* i) = 0 when i is an integer.
So you can just multiply f(x,y) by -1^(x+y)

```
rows[row][col] = std::complex<double>(padded_f.at<float>(row, col) *
std::pow(-1, row+col), 0);
```

UI

A new operation - the Fourier Transform - is added in the first tab. When the operation is run, The original image and Fourier Transform as seen in the Operands View. Toggle the checkbox `Operands/Output Toggle` to switch to Output view, which shows the ifft result.

The fourier transform image is reorganized to center the upper left corner, and tile the rest of the image
so that the frequencies it represents are mirrored about both axes.

Analysis

Included in this folder is an excel sheet with run times for the implemented DFT functions, In microseconds, where for each image measurements are taken while scaling it down by 0.5.

In the excel files, there are graphs

PP2 Questions

1. Compare and contrast the two approaches for resizing images (resizing to a common size, vs resizing to the larger image size) when performing an image operation that involves more than one image. Which approach is suitable for a medical imaging application like an MRI? Which approach is suitable for a simulation application like finite element modeling on the GPU?

Resizing to the larger image size will cause no loss of data. It may however introduce artifacts in the output image, because we are introducing the blocky patterns of an
image that is bigger than its original resolution.

For medical imaging, we probably don't want to introduce any artifacts, so we probably
want to resize to a common size.

When we are using images on the GPU, especially when they are effectively being used
as Look Up Tables, we want no loss of data - we want the highest resolution data we
can get (the rendering API will handle interpolating and Level Of Detail for us anyway). So, in
this case,
we probably want to size to the highest size.

2. We believe that the histogram equalization process always gives a better-contrast image than the original. We also believe that it is a non-destructive process i.e. it will not reduce the contrast in an already “good-enough” image. Show that the preceding statements are FALSE with a counter example. Use a 2x2 image to demonstrate your answer.

0 0
0 8
L = 8

```
r    p_cuml    s
-----------------
0   0.75       6
8   1.0        8
```

The range was 8, but is now just 2.

Histograms (PP2)

Filling the gap in G_inv

In order to perform histogram patching, we calculate the cumulative T table for texture B,
which is referred to as G in the book. Then, we calculate the inverse function - all the
[key, value] pairs in G become [value, key] pairs in G_inv.

At the end of this process, there are a lot of intensities which are missing as values from G_inv.

If nothing is done, there is noise scattered throughout the result image.

In this app, we interpolate linearly between values in the lookup. After computing G_inv,
we loop through it and replace any zeroes like this:

```
// find the nearest mapping, and interpolate
//| 4 | 0 | 0 | 0 | 0 | 9 |
//  0   1  ^2   3   4   5
// range = 5-0=5,
// weigh non-0 neighbors by distance from i: (1-2/5)*4 + (1-3/5) * 9
```

Filters

This is a UI for blur and smoothing

Technologies Being Used

This is a CMake project, using [vcpkg](microsoft/vcpkg: C++ Library Manager for Windows, Linux, and MacOS (github.com)) for dependency management. See `vcpkg.json` for a list of libraries that this project depends on.

Building

`Build.bat` in the root directory should produce a `build/dip.exe` program.

This script:

- Downloads and bootstraps vcpkg into /bin/vcpkg
- Runs `vcvarsall.bat` on a detected visual studio install, thereby setting the environment for CMake. Then, hopefully CMake will pick up the right compiler.
- Performs CMake configure step, which should use the vcpkg toolchain and install all the dependencies from the internet. **This may take a while - it's going to build opencv and glfw**.
- Runs build
-

Prerequisites

- Visual studio version 2022 or 2019 with C++ features installed.
- C++17
- `Build.bat` tries to use 2022, 2019 or 2017. Anything older assuming won't work.

Features

The application state has 3 images: A, B, OUTPUT. Operations require either 1 or 2 operands.
For one operand operations, A is the source.
Results are output always to OUTPUT.

There's going to be a Operands and Output views, where operands shows a split screen view of A and B.

The output view will show OUTPUT.

The range was 8 at first, but after histogram eq, it is 2.

(PP1)

Operations

This will have various controls for performing operations for A and B, and writing the result to output.
It will also Have controls for switching views.

Console

This is a custom console with a bunch of commands available.
`cd`, `ls` are implemented. Tab completion has very basic implementation.

```
[load|save] <ppm_path>"
[add|sum|mul] -i <ppm_path1> <ppm_path2> -i <out_ppm_path>
inv -i <ppm_path1> -o <out_ppm_path>
log -i <ppm_path1> -o <out_ppm_path> -c <const> -b <base>
pow -i <ppm_path1> -o <out_ppm_path> -c <const> -gamma <gamma>
```

Platforms

Windows.
Was developed using 2022 Visual Studio toolkit, tested on 2019.

What needs to be ported?

`/src/os/browse_dialog.cpp`

Implementation for the native file browse dialog on windows.

It's possible (looks like tricky on osx) to add implementations for osx/linux, or to just use a library from github.

A basic file browser can also be implemented in IMGui.