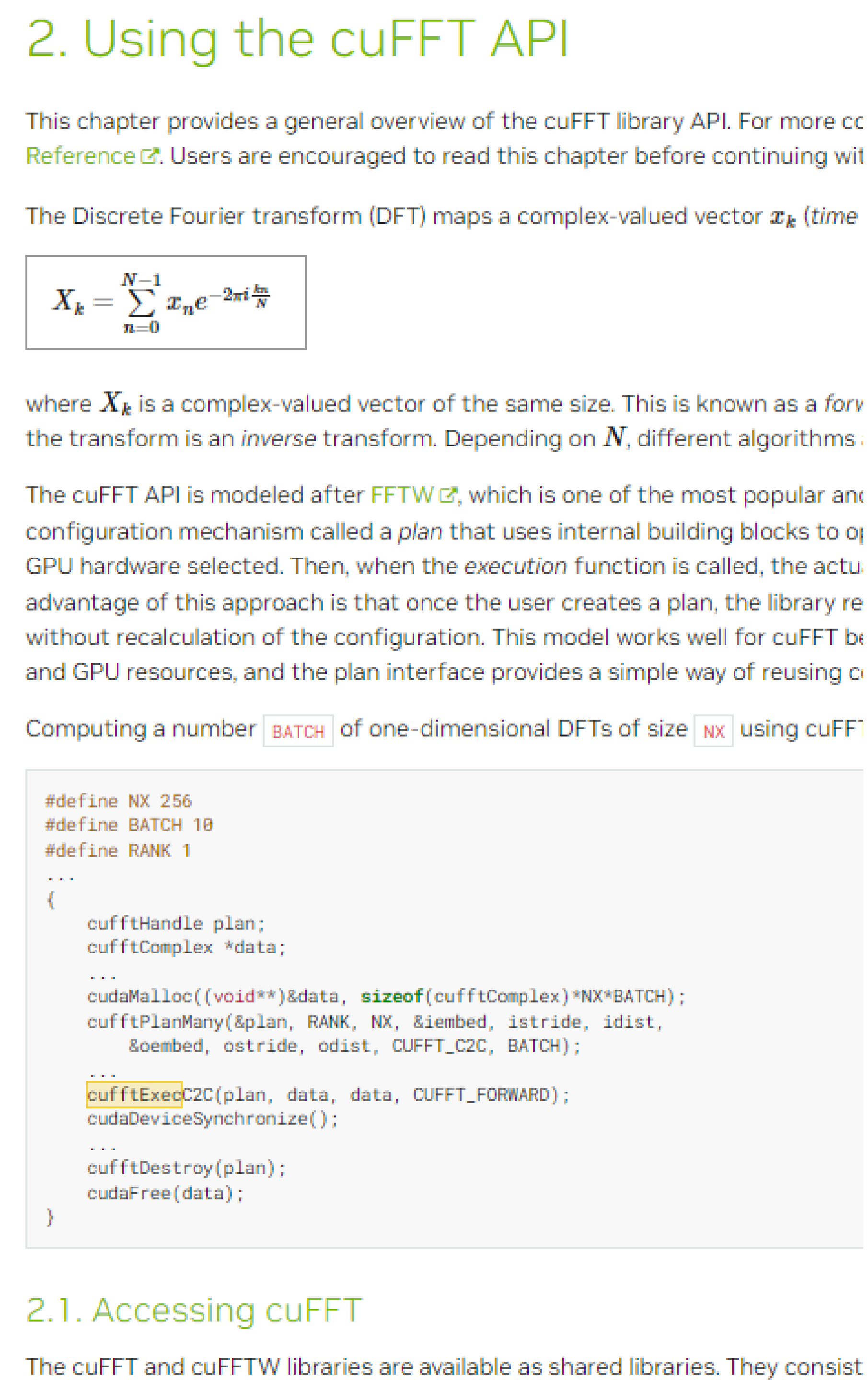
FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

```
NppiSize oMaskSize = {5, 5};

NppiSize oSrcSize = {(int)oDeviceSrc.width(), (int)oDeviceSrc.height()};
NppiPoint oSrcOffset = {0, 0};

// create struct with ROI size
NppiSize oSizeROI = {(int)oDeviceSrc.width(), (int)oDeviceSrc.height()};
// allocate device image of appropriately reduced size
npp::ImageNPP_8u_C1 oDeviceDst(oSizeROI.width, oSizeROI.height);
// set anchor point inside the mask to (oMaskSize.width / 2,
// oMaskSize.height / 2) It should round down when odd
NppiPoint oAnchor = {oMaskSize.width / 2, oMaskSize.height / 2};

// run box filter
NPP_CHECK_NPP(nppiFilterBoxBorder_8u_C1R(
    oDeviceSrc.data(), oDeviceSrc.pitch(), oSrcSize, oSrcOffset,
    oDeviceDst.data(), oDeviceDst.pitch(), oSizeROI, oMaskSize, oAnchor,
    NPP_BORDER_REPLICATE));

// declare a host image for the result
npp::ImageCPU_8u_C1 oHostDst(oDeviceDst.size());
// and copy the device result data into it
oDeviceDst.copyTo(oHostDst.data(), oHostDst.pitch());

saveImage(sResultFilename, oHostDst);
std::cout << "Saved image: " << sResultFilename << std::endl;

// verified
// Redo Lena *.png from PGMs
cv::Mat img_Lena_orig = imread("Lena.pgm", cv::IMREAD_UNCHANGED);
cv::imwrite("./Lena_orig.png", img_Lena_orig);
cv::Mat img_Lena_boxFilter = imread("Lena_boxFilter.pgm", cv::IMREAD_UNCHANGED);
cv::imwrite("./Lena_postBox.png", img_Lena_boxFilter);

nppiFree(oDeviceSrc.data());
nppiFree(oDeviceDst.data());

// ************ FFT processing *************************
```



2. Using the cuFFT API

This chapter provides a general overview of the cuFFT library API. For more co
Reference. Users are encouraged to read this chapter before continuing wit

The Discrete Fourier transform (DFT) maps a complex-valued vector x_k (time

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i \frac{kn}{N}}$$

where X_k is a complex-valued vector of the same size. This is known as a forv
the transform is an *inverse* transform. Depending on N, different algorithms

The cuFFT API is modeled after FFTW, which is one of the most popular and
configuration mechanism called a *plan* that uses internal building blocks to o
GPU hardware selected. Then, when the *execution* function is called, the actu
advantage of this approach is that once the user creates a plan, the library re
without recalculation of the configuration. This model works well for cuFFT be
and GPU resources, and the plan interface provides a simple way of reusing c

Computing a number `BATCH` of one-dimensional DFTs of size `NX` using cuFF

```
#define NX 256
#define BATCH 10
#define RANK 1
...
{
    cufftHandle plan;
    cufftComplex *data;
    ...
    cudaMalloc((void**)&data, sizeof(cufftComplex)*NX*BATCH);
    cufftPlanMany(&plan, RANK, NX, &iembed, istride, idist,
        &oembed, ostride, odist, CUFFT_C2C, BATCH);
    ...
    cufftExecC2C(plan, data, data, CUFFT_FORWARD);
    cudaDeviceSynchronize();
    ...
    cufftDestroy(plan);
    cudaFree(data);
}
```

2.1. Accessing cuFFT

The cuFFT and cuFFTW libraries are available as shared libraries. They consist

OUTLINE

- Background of Fast Fourier Transform (FFT)
- FFT implementation in cuda code
- Effect of hf filter on lena.pgm
- FFT of vertical lines at fixed pitch
- Summary
- APPENDIX: Makefile mods

EFFECT OF HF FILTER IN LENA.PGM

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

FAST FOURIER TRANSFORM BACKGROIUND

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

FFT BACKGROUND

- FFT of the image produces a complex frequency spectrum
- Possible to take the inverse FFT of the image frequency to get back the original image
- Can filter the frequency spectrum to see effect on images.
- If the image has only 1 underlying pitch, should only be 1 non zero frequency in frequency spectrum

$$f(x) = \int_{-\infty}^{\infty} F(k)\, e^{2\pi i k x}\, dk$$

$$F(k) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i k x}\, dx.$$

mathworld.wolfram.com/FourierTransform.html

Fast Fourier transform

Let $x_0, \ldots, x_{n-1}$ be complex numbers. The DFT is defined

$$X_k = \sum_{m=0}^{n-1} x_m e^{-i2\pi km/n} \qquad k = 0, \ldots, n-1,$$

where $e^{i2\pi/n}$ is a primitive n'th root of 1.

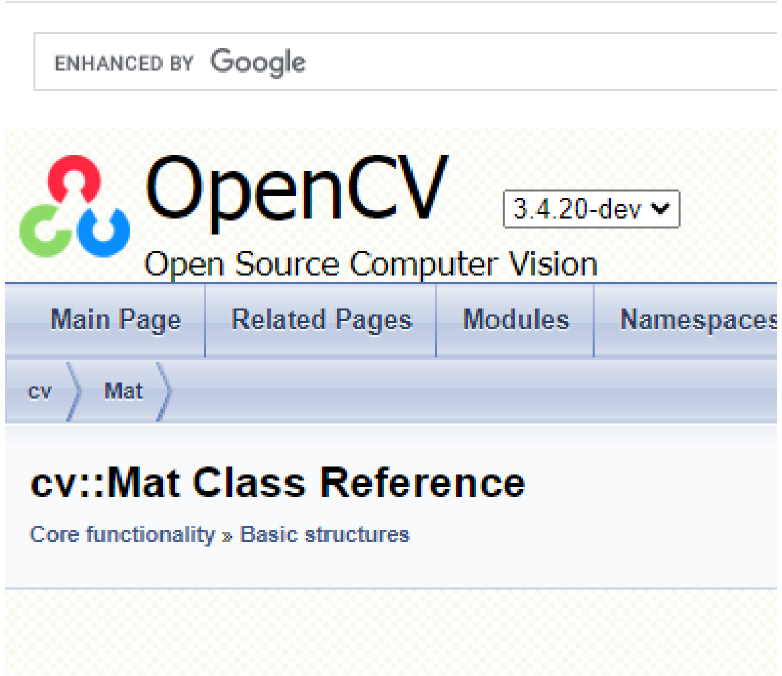
en.wikipedia.org/wiki/Fast_Fourier_transform

FFT CUDA CODE

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

FFT CUDA CODE FOR LENA.PGM

- Added FFT processing to existing boxFilterNPP.cpp
- ChatGPT hallucinated quite a few times in creating NPP versions of cuFFT that did not exist.
- Used OpenCV cv:: functions to load lena.pgm to convert from *.pgm into *cufftReal format
- Performed FFT of png composed vertical lines pitch = 20.
  - Actually sin^2(2*!PI*x/20)
- Helper functions for creating *.png of frequency spectrums
- Helper functions to create *.png of vertical sin^2() lines of fixed pitch



```cpp
// ************ FFT processing *************************

// Load the original image using OpenCV
cv::Mat originalImage = cv::imread("Lena_orig.png", cv::IMREAD_GRAYSCALE);
if (originalImage.empty())
{
  std::cerr << "Error: Failed to load original image." << std::endl;
  return 1;
}

// Get image dimensions
int width = originalImage.cols;
int height = originalImage.rows;

// Allocate memory for input and output data on the GPU
cufftReal *d_inputData;
cufftComplex *d_outputData;
cudaMalloc(&d_inputData, width * height * sizeof(cufftReal));
cudaMalloc(&d_outputData, (width / 2 + 1) * height * sizeof(cufftComplex)); // Only store half of the complex numbers for real input

// Convert image data to cufftReal format and copy to GPU memory
cufftReal *h_imageData = new cufftReal[width * height];
for (int i = 0; i < height; ++i)
{
  for (int j = 0; j < width; ++j)
  {
    h_imageData[i * width + j] = static_cast<cufftReal>(originalImage.at<unsigned char>(i, j));
  }
}
cudaMemcpy(d_inputData, h_imageData, width * height * sizeof(cufftReal), cudaMemcpyHostToDevice);
delete[] h_imageData;
```

```cpp
    // Create a forward FFT plan
    cufftHandle forwardPlan;
    cufftPlan2d(&forwardPlan, height, width, CUFFT_R2C);

    // Execute forward FFT
    cufftExecR2C(forwardPlan, d_inputData, d_outputData);

    // Copy the FFT data back to host memory
    cufftComplex *h_outputDataUnblocked = new cufftComplex[(width / 2 + 1) * height];
    cudaMemcpy(h_outputDataUnblocked, d_outputData, (width / 2 + 1) * height * sizeof(cufftComplex), cudaMemcpyDeviceToHost);

    // void saveComplexSquares(const cufftComplex *h_outputData, int width, int height, const std::string &name)
    saveComplexSquares(h_outputDataUnblocked, (width / 2 + 1), height, "Freq_Lena_Unblocked");

    // Define the bands to block (right and bottom)
    int blockWidth = static_cast<int>(0.1 * width);     // 10% of width
    int blockHeight = static_cast<int>(0.1 * height);   // 10% of height

    // Allocate memory for blocked output data on the host
    cufftComplex *h_outputDataBlocked = new cufftComplex[(width / 2 + 1) * height];
    cudaMemcpy(h_outputDataBlocked, h_outputDataUnblocked, (width / 2 + 1) * height * sizeof(cufftComplex), cudaMemcpyHostToHost);

    // Apply block filter on the host
    blockFrequenciesHost(h_outputDataBlocked, width / 2 + 1, height, blockWidth, blockHeight);

    // Copy the filtered FFT data back to device memory
    cudaMemcpy(d_outputData, h_outputDataBlocked, (width / 2 + 1) * height * sizeof(cufftComplex), cudaMemcpyHostToDevice);

    // Perform inverse FFT on the blocked data
    cufftHandle inversePlan;
    cufftPlan2d(&inversePlan, height, width, CUFFT_C2R);
    cufftExecC2R(inversePlan, d_outputData, d_inputData);

    // Copy reconstructed image data back to host memory
    cufftReal *h_reconstructedDataBlocked = new cufftReal[width * height];
    cudaMemcpy(h_reconstructedDataBlocked, d_inputData, width * height * sizeof(cufftReal), cudaMemcpyDeviceToHost);
```

```
    // Normalize the reconstructed image to the range [0, 255]
    cv::Mat reconstructedImageBlocked(height, width, CV_32FC1, h_reconstructedDataBlocked);
    cv::normalize(reconstructedImageBlocked, reconstructedImageBlocked, 0, 255, cv::NORM_MINMAX, CV_8UC1);

    // Save the blocked reconstructed image as a PNG file
    cv::imwrite("reconstructed_image_blocked.png", reconstructedImageBlocked);

    // void createVerticalSineSquared(const std::string &name, int period)
    //createVerticalSineSquared("sin_Per_20", 20);



    // Clean up
    delete[] h_outputDataUnblocked;
    delete[] h_outputDataBlocked;
    delete[] h_reconstructedDataBlocked;
    cufftDestroy(forwardPlan);
    cufftDestroy(inversePlan);
    cudaFree(d_inputData);
    cudaFree(d_outputData);

    // repeat above analysis for periodic vertical sin bars

    // Load the original image using OpenCV
    cv::Mat originalImage2 = cv::imread("sin_Per_20.png", cv::IMREAD_GRAYSCALE);
    if (originalImage2.empty())
    {
      std::cerr << "Error: Failed to load original image." << std::endl;
      return 1;
    }

    // Get image dimensions
    int width2 = originalImage.cols;
    int height2 = originalImage.rows;

```

```
// Allocate memory for input and output data on the GPU
cufftReal *d_inputData2;
cufftComplex *d_outputData2;
cudaMalloc(&d_inputData2, width2 * height2 * sizeof(cufftReal));
cudaMalloc(&d_outputData2, (width2 / 2 + 1) * height2 * sizeof(cufftComplex)); // Only store half of the complex numbers for real input

// Convert image data to cufftReal format and copy to GPU memory
cufftReal *h_imageData2 = new cufftReal[width2 * height2];
for (int i = 0; i < height2; ++i)
{
  for (int j = 0; j < width2; ++j)
  {
    h_imageData2[i * width2 + j] = static_cast<cufftReal>(originalImage2.at<unsigned char>(i, j));
  }
}
cudaMemcpy(d_inputData2, h_imageData2, width2 * height2 * sizeof(cufftReal), cudaMemcpyHostToDevice);
delete[] h_imageData2;

// Create a forward FFT plan
cufftHandle forwardPlan2;
cufftPlan2d(&forwardPlan2, height2, width2, CUFFT_R2C);

// Execute forward FFT
cufftExecR2C(forwardPlan2, d_inputData2, d_outputData2);

// Copy the FFT data back to host memory
cufftComplex *h_outputDataUnblocked2 = new cufftComplex[(width / 2 + 1) * height];
cudaMemcpy(h_outputDataUnblocked2, d_outputData2, (width2 / 2 + 1) * height2 * sizeof(cufftComplex), cudaMemcpyDeviceToHost);

// void saveComplexSquares(const cufftComplex *h_outputData, int width, int height, const std::string &name)
saveComplexSquares(h_outputDataUnblocked2, (width2 / 2 + 1), height2, "FreqSpect_Sin2_P40");

// ************end FFT processing ******************
```

FFT CUDA CODELENA.PGM BLOCKED AND UNBLOCKED

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

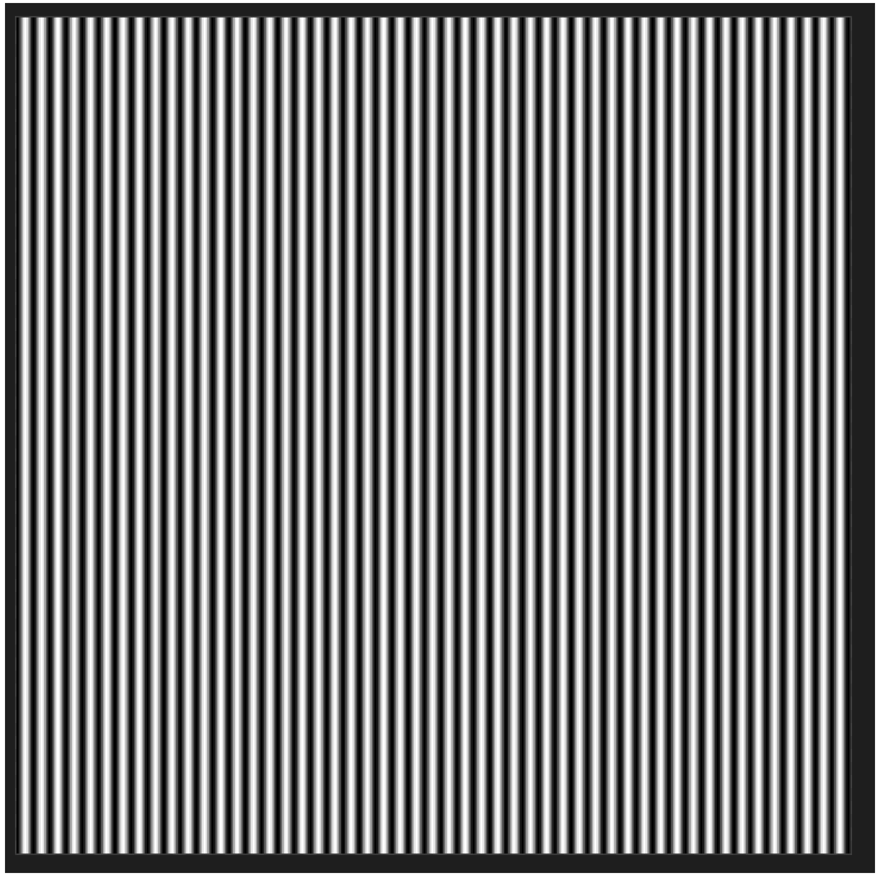
LENA IMAGES ORIGINAL AND BLOCKED 10%OF HIGHEST FREQUENCIES BLOCKED

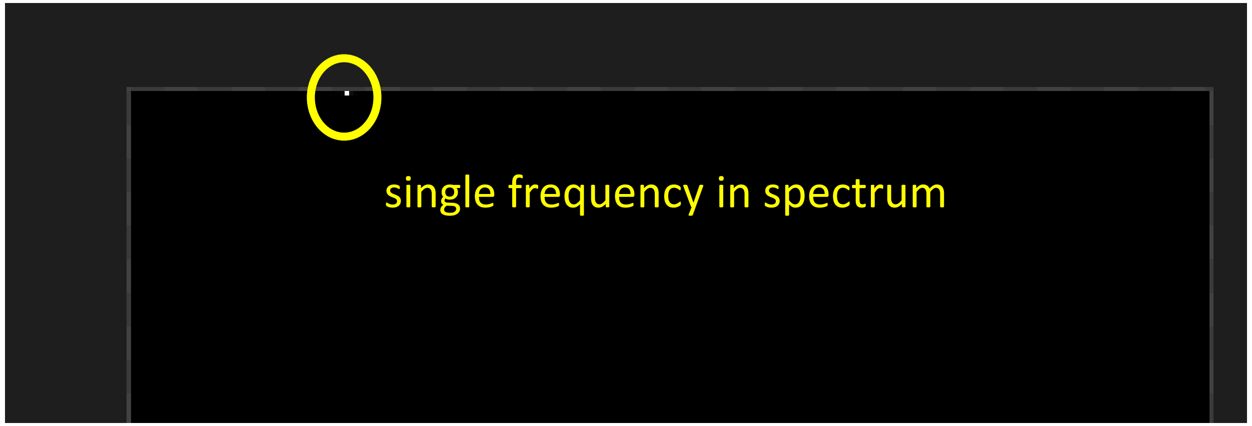
```
// Define the bands to block (right and bottom)
int blockWidth = static_cast<int>(0.1 * width);     // 10% of width
int blockHeight = static_cast<int>(0.1 * height);   // 10% of height
```

FFT OF FIXED PITCH VERTICAL LINES

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

FFT OF VERTICAL LINES AT PITCH=20
IMAGE IS 512 X 512





SUMMARY

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

SUMMARY

- Successfully added FFT functionality to the existing boxFilterNPP.cpp program
- FFT followed by inverse FFT successfully recovers the original image
- When the top 10% of the frequency spectrum was blocked, lena.png was visibly modified
- When the fft of vertical lines was taken, the resulting frequency spectrum showed only a single frequency when the frequency spectrum was p[lotted as a *.png

APPENDIX: MAKEFILE MODS

FFT PROCESSING ADDED TO BOXFILTERNPP.CPP

APPENDIX: MAKEFILE MODS NEEDED

- It was challenging to bring in new modules into boxFilterNPP.cpp
- Needed to make modifications to access cv:: libraries
- Open big problem was opencv4 vs opencv2
- Tremendous amount of trial and error

```
# Common includes and paths for CUDA
INCLUDES  := -L/usr/lib/x86_64-linux-gnu -I../Common -I/usr/include/opencv4 -lopencv_core -lopencv_imgcodecs
LIBRARIES :=
```

```
ALL_CCFLAGS += --threads 0

INCLUDES += -I../Common/UtilNPP

LIBRARIES += -lnppisu_static -lnppif_static -lnppc_static -lculibos -lfreeimage -lopencv_core -lopencv_imgcodecs -lcufft
```

THANK YOU

Peter Brooker