Programming Assignment: 2D Convolution

ECE 6397
Electrical and Computer Engineering
University of Houston
Dr. David Mayerich
2D Convolution

• Given a two-dimensional function $I$ and a convolution kernel $K$:

$$I(x, y) * K = \int_{u=-\infty}^{\infty} \int_{v=-\infty}^{\infty} K(u, v) \cdot I(x - u, y - v) du \, dv$$

• For an image $I \in \mathbb{R}^{m \times n}$ and kernel $K \in \mathbb{R}^{k \times k}$ made of discrete samples:

$$I(x, y) * K = \sum_{u=-\frac{k}{2}}^{\frac{k}{2}} \sum_{v=-\frac{k}{2}}^{\frac{k}{2}} K[u, v] \cdot I(x - u, y - v)$$
Image Convolution

\[ I(x, y) = \]

\[ K(x, y) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = \]

\[ I(x, y) * K = \]
Convolution between two functions is equivalent to multiplication of their Fourier transforms:

\[ I(x, y) \ast K = \mathcal{F}(I)(f_x, f_y) \cdot \mathcal{F}(K) \]

- Use cuFFT (nVidia library) to perform a forward transform of \( I \) and \( K \)
  - Write a kernel to perform the multiplication

- A DFFT requires \( O(n \log n) \)
  - where \( n \) is the number of elements in the signal

- A convolution requires \( O(nm) \)
  - where \( m \) is the number of elements in the kernel

![GFLOPS used for a 1D cuFFT as a function of data size](image.png)
Discrete Convolution

\[ y_0 \quad y_1 \quad \cdots \quad y_{n-2} \quad y_{n-1} \]

\[ x_0 \quad x_1 \quad \cdots \quad x_{m-2} \quad x_{m-1} \]
Separable Convolution

• A separable filter can be written as a product of two filters
  \[ K = K_0 \cdot K_1 \]

• A Gaussian filter is separable because

\[
\frac{1}{2\sigma^2\pi} e^{-\frac{(x-\mu)^2 + (y-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \cdot \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}
\]
Discrete Convolution

\[
\begin{array}{cccc}
  y_0 & y_1 & \cdots & y_{n-1} \\
  u_0 & \cdots & u_{k-1} \\
\end{array}
\]

\[
\begin{array}{cccc}
  v_0 & v_1 & \cdots & v_{k-1} \\
\end{array}
\]

\[
\begin{array}{cccc}
  x_0 & x_1 & \cdots & x_{m-1} \\
\end{array}
\]
Implementation Details

• Convolution is highly data parallel
  • The result of each pixel \((x, y)\) in the output image is independent of all other values
  • Each output pixel requires implementing the same sequence of instructions

• Memory latency
  • Adjacent threads can access adjacent data
  • Warps will access neighboring pixel values in \(I(x, y)\)
  • Warps will access the same pixel values in \(K(u, v)\)
  • *Think about how you’ll store and retrieve your data from the GPU*
Programming Assignment Input

• Implement a Gaussian convolution program that’s called using the following signature:

  $\gg \text{convolve imagename.ppm \ 40}$

  • the first argument is the name of the image file (all images will be *.ppm)
  • the second argument is the standard deviation of the Gaussian kernel ($\sigma$)

• Images will be large – several are available on the website

![Sombrero Galaxy](image1.png) Sombrero Galaxy 11472 × 6429
![Whirlpool Galaxy](image2.png) Whirlpool Galaxy 11477 × 7965
![Hereford Cathedral](image3.png) Hereford Cathedral 9999 × 7435
![Bansberia Royal Estate](image4.png) Bansberia Royal Estate 12726 × 5604
Arguments

• Arguments are passed as text by the operating system to the `main(...)` function:

```cpp
#include <string>

int main(int argc, char* argv[]){
    if(argc != 3) //there should be three arguments
        return 1; //exit and return an error
    std::string filename(argv[1]);
    int sigma = atoi(argv[2]);
    ...
}
```

• remember: the first argument is the command (executable name)
• all arguments are stored as NULL terminated character strings
Netpbm Images – *.ppm

- Images use the Netpbm format: [https://en.wikipedia.org/wiki/Netpbm_format](https://en.wikipedia.org/wiki/Netpbm_format)
- The first line of the file is a line of text saying “P6”
  - indicates that this is a binary ppm (color image) file
- The second line of the file is a comment starting with “#”
  - in general, there can be any number of comment lines – test the first character
- The next line of the file has two integers (in ASCII)
  - these specify the width and height of the image
- The last text line of the file has one integer (in ASCII)
  - this specifies the maximum intensity value – 255 for all sample images
- The rest of the file stores pixel data in binary
  - Each pixel is in lexicographic (row-major) order
  - The data type for each value is an unsigned char
  - The data is stored in RGB triples
PPM Data Dump

• Data dump from [https://hexed.it/](https://hexed.it/):

```
00006600 50 36 0A 23 20 43 52 45 41 54 4F 52 3A 20 47 49
00006610 4D 50 20 50 4E 45 4C 45 43 48 49 46 46 72 61 6C 74
00006620 76 31 31 20 44 4F 43 48 49 46 46 72 61 6C 74 20 3A
00006630 31 31 20 32 30 30 30 30 30 30 30 30 30 30 30 30
00006640 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00006650 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006660 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006670 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006680 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006690 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000066A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000066B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000066C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000066D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000066E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000066F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006700 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006710 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006720 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006730 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006740 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006750 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00006760 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

- specifies binary PPM file
- comment: (I made this file with GIMP)
- X and Y dimensions
- maximum value: 0.255

Pixel data (in this image all pixels are red)
Displaying PPM files

• PPM files can be viewed, saved, and converted with Honeyview or GIMP:
  
  https://www.gimp.org/
  https://en.bandisoft.com/honeyview/

• I recommend making some small images to test your code out with
Programming Assignment Output

• Your code will perform two functions
  • Implement a CPU-based Gaussian filter, saving the image when done
  • Implement a GPU-based Gaussian filter, saving the image when done
  • Time both algorithms and display the timing results

• Implementation specifics:
  • The standard deviation will be given in pixels (and will be an integral type)
  • Each element of the convolution kernel should be a floating point value
  • The kernel will be calculated as a normal distribution
  • Make sure that your kernel keeps 99% of the Gaussian for accuracy
Calculating the size of your kernel

• How big will the kernel be with $\sigma = 40$ pixels?
  • $99.7\%$ of a normal Gaussian falls within 3 standard deviations from the mean
  • The mean in this case is zero ($\mu = 0$)

$$k = 2 \cdot 3 \cdot \sigma = 6\sigma$$

• note: it’s easier to have an odd kernel

```c
int k = 6 * sigma; //calculate k
if(k % 2 == 0) k++; //make sure k is odd

//allocate space for the Gaussian kernel
float* K = (float*) malloc(k * sizeof(float));
for(int i = 0; i < k; i++)
    K[i] = ... //evaluate the Gaussian

... //perform the convolution
```