IMAGE CONVOLUTION

LEARNING OBJECTIVES

- Learn about image convolutions and what makes them a good problem for solving on a GPU
- Learn what a naive image convolution may look like

IMAGE CONVOLUTION

Over the next few lectures we will be looking at some common GPU optimizations with an image convolution as the motivational example.

- A good problem to solve on a GPU.
- Can take advantage of a number of common optimizations.
- Convolution is a very powerful algorithm with many applications.
- Deep neural networks.
- Image processing.
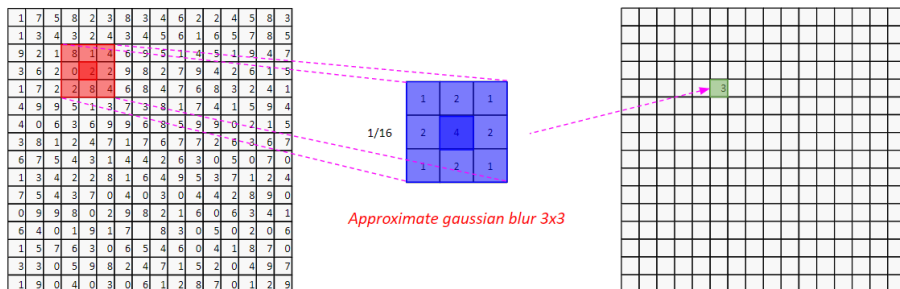
WHY ARE IMAGE CONVOLUTIONS GOOD ON A GPU?

- The algorithm is **embarrassingly parallel**.
- Each work-item in the computation can be calculated entirely independently.
- The algorithm is computation heavy.
- A large number of operations are performed for each work-item in the computation, particularly when using large filters.
- The algorithm requires a large bandwidth.
- A lot of data must be passed through the GPU to process an image, particularly if the image is very high resolution.

IMAGE CONVOLUTION DEFINITION

$$G = h \otimes F \qquad G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] F[i+u, j+v]$$

1/16

Approximate gaussian blur 3x3

IMAGE CONVOLUTION DEFINITION

$$G = h \otimes F \qquad G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v]F[i+u,j+v]$$

Approximate gaussian blur 3x3

- A filter of a given size is applied as a stencil to the position of each pixel in the input image.
- Each pixel covered by the filter is then multiples with the corresponding element in the filter.
- The result of these multiplications is then summed to give the resulting output pixel.
- Here we have a 3x3 gaussian blur approximation as an example.

IMAGE CONVOLUTION EXAMPLE

IMAGE CONVOLUTION DATA FLOW
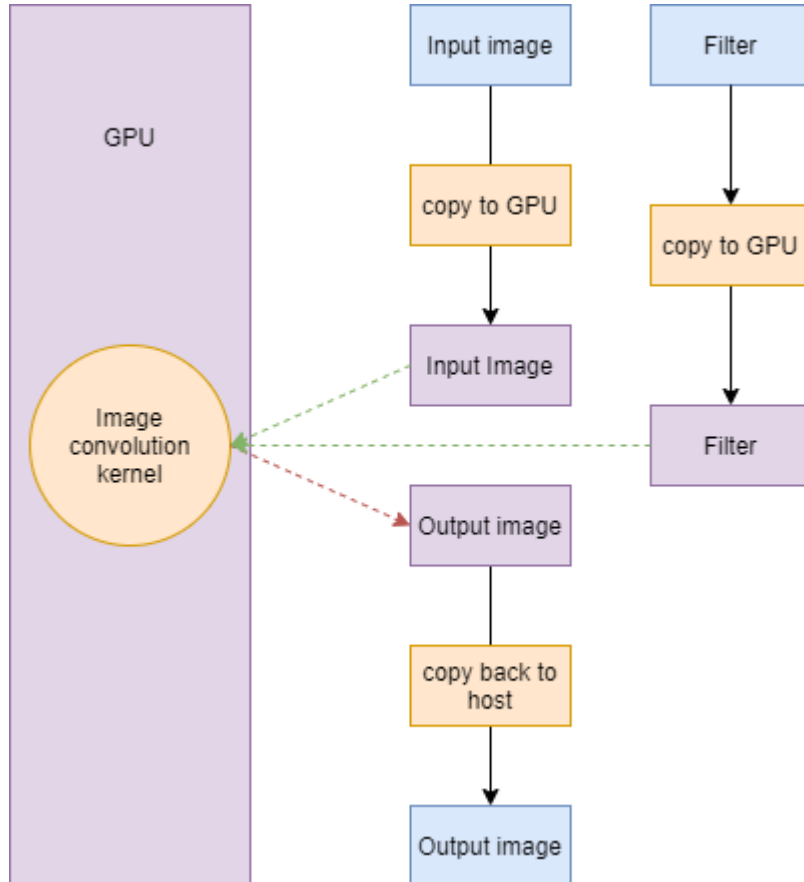
- We have a single kernel function.
- It must read from the input image data and writes to the output image data.
- It must also read from the filter.
- The input image data and the filter don't need to be copied back to the host.
- The output image data can be uninitialized.

IMPLEMENTATION

- We provide a naive implementation of a SYCL application which implements the image convolution algorithm.
- This will be the basis for optimization in later lectures and exercises.
- The implementation uses the stb image library to allow us to visualize our results.
- The implementation also uses a benchmark function to allow us to measure the performance as we make optimizations.

REFERENCE IMAGE

- We provide a reference image to use in the exercise.
- This is in Code_Exercises/Images
- This image is a 512x512 RGBA png.
- Feel free to use your own image but we recommend keeping to this format.

INPUT/OUTPUT IMAGE LOCATIONS

```
auto inputImageFile = "../Code_Exercises/Images/dogs.png";
auto outputImageFile = ../Code_Exercises/Images/blurred_dogs.png";
```

- The reference code and the solutions to the remaining exercises use these strings to refernce the location of the input and output image.
- Before compiling these you will have to update this to point to the image in the development environment.

codeplay®

CONVOLUTION FILTERS

```
auto filter = util::generate_filter(util::filter_type filterType, int width);
```

- The utility for generating the filter data takes a `filter_type` enum which can be either `identity` or `blur` and a width.
- Feel free to experiment with different variations.
- Note that the filter width should always be an odd value.
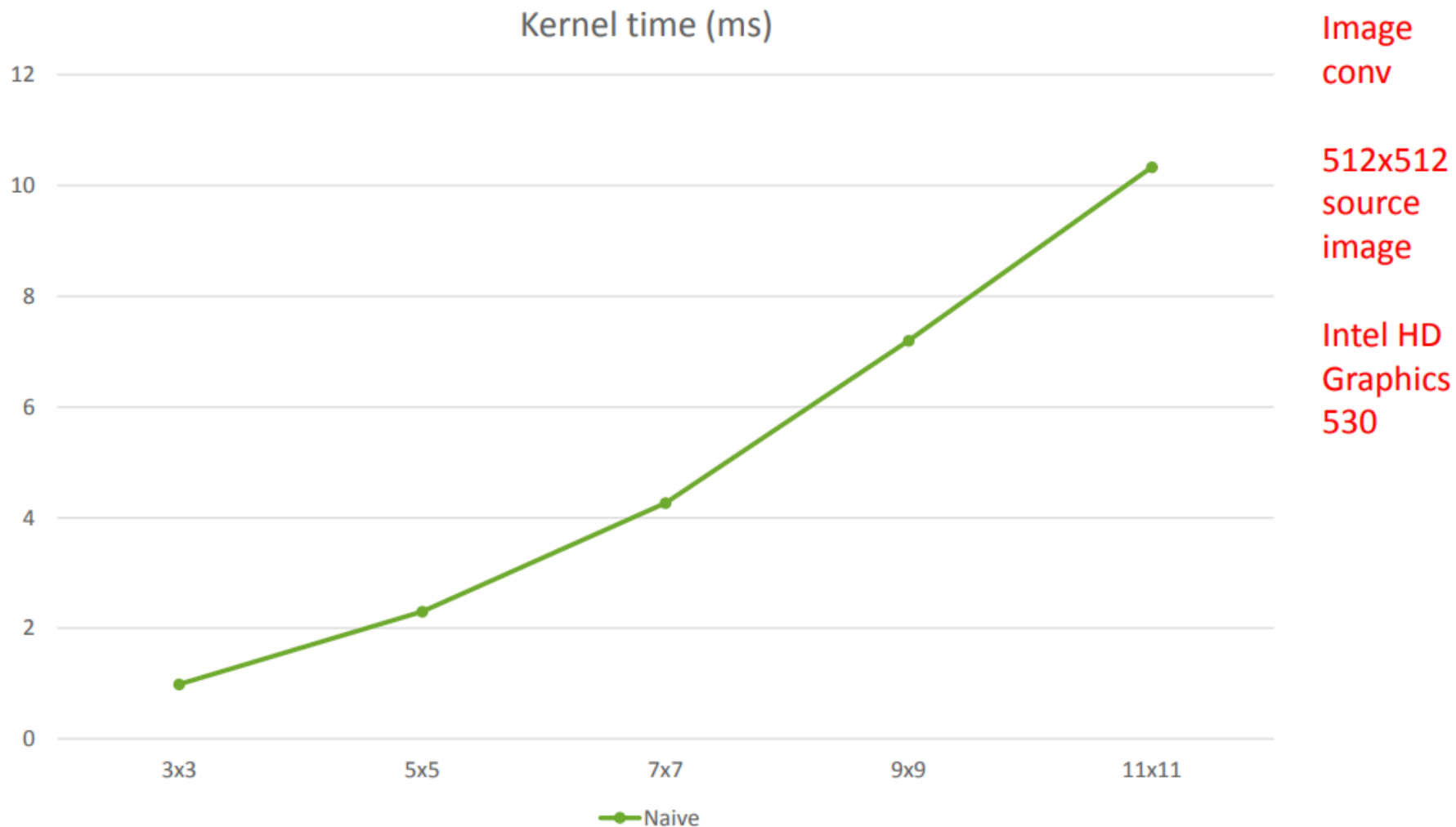
QUESTIONS

Code_Exercises/Exercise_15_Image_Convolution/reference

Take a look at the naive image convolution code provided and familiarize yourself with it.