Convolution

Objective

The lab's objective is to implement a tiled image convolution using both shared and constant memory. We will have a constant 5x5 convolution mask, but will have arbitrarily sized image (assume the image dimensions are greater than 5x5 for this Lab).

To use the constant memory for the convolution mask, you can first transfer the mask data to the device. Consider the case where the pointer to the device array for the mask is named M. You can use `const float * __restrict__ M` as one of the parameters during your kernel launch. This informs the compiler that the contents of the mask array are constants and will only be accessed through pointer variable `M`. This will enable the compiler to place the data into constant memory and allow the SM hardware to aggressively cache the mask data at runtime.

Convolution is used in many fields, such as image processing for image filtering. A standard image convolution formula for a 5x5 convolution filter `M` with an Image `I` is:

$$P_{i,j,c} = \sum_{x=-2}^{2} \sum_{y=-2}^{2} I_{i+x,j+y,c} * M_{x,y}$$

where $P_{i,j,c}$ is the output pixel at position `i,j` in channel `c`, $I_{i,j,c}$ is the input pixel at `i,j` in channel `c` (the number of channels will always be 3 for this MP corresponding to the RGB values), and $M_{x,y}$ is the mask at position `x,y`.

Input Data

The input is an interleaved image of `height x width x channels`. By interleaved, we mean that the element `I[y][x]` contains three values representing the RGB channels. This means that to index a particular element's value, you will have to do something like:

```
index = (yIndex*width + xIndex)*channels + channelIndex;
```

For this assignment, the channel index is 0 for R, 1 for G, and 2 for B. So, to access the G value of `I[y][x]`, you should use the linearized expression `I[(yIndex*width+xIndex)*channels + 1]`.

For simplicity, you can assume that `channels` is always set to `3`.

Instructions

Edit the code in the code tab to perform the following:

- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel

- copy results from device to host
- deallocate device memory
- implement the tiled 2D convolution kernel with adjustments for channels
- use shared memory to reduce the number of global accesses, handle the boundary conditions in when loading input list elements into the shared memory

Pseudo Code

A sequential pseudo code would look something like this:

```
maskWidth := 5
maskRadius := maskWidth/2 # this is integer division, so the result is 2
for i from 0 to height do
  for j from 0 to width do
    for k from 0 to channels
      accum := 0
      for y from -maskRadius to maskRadius do
        for x from -maskRadius to maskRadius do
          xOffset := j + x
          yOffset := i + y
          if xOffset >= 0 && xOffset < width &&
             yOffset >= 0 && yOffset < height then
            imagePixel := I[(yOffset * width + xOffset) * channels + k]
            maskValue := K[(y+maskRadius)*maskWidth+x+maskRadius]
            accum += imagePixel * maskValue
          end
        end
      end
      # pixels are in the range of 0 to 1
      P[(i * width + j)*channels + k] = clamp(accum, 0, 1)
    end
  end
end
```

where `clamp` is defined as

```
def clamp(x, lower, upper)
  return min(max(x, lower), upper)
end
```