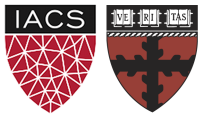
CS205, Fall 2012
Computing Foundations for Computational Science
Hanspeter Pfister, Cris Cecka



HARVARD
School of Engineering
and Applied Sciences

Homework 5 – Due November, 21st, 2012

You just got hired to work on resizing videos for Youtube. Excited about that, but lacking the fundamentals of video processing, you decide to grab a book and study.... but then, your favorite class comes through with a homework that will definitely help and let you save tons of time!

In this homework you will manipulate videos using multiple GPUs. As such, you will combine your knowledge of CUDA programming on the GPUs with the power of MPI parallelization. You will also use Cheetah metaprogramming which allows you to generate code on the fly.

Download the code for HW5 here.

or

`wget http://www.cs205.org/resources/hw5.zip`

Problem 1 - Video Seam Carving

Your task is to resize images and videos. The problem is as follows: given an image (or video) that has been shot for a given aspect ratio (for instance, 16:9), resize this image so that it fits another aspect ratio (4:3).
Two solutions are trivial: either crop the image, or stretch it. In the first case, this leads to loss in information, while in the second case, this can lead to important deformations (see Fig.). We are rather interested in a third solution that proposes a tradeoff. If you try to stretch down an image horizontally, it will find a seam – a vertical 1-pixel wide path – in the image that would go unnoticed if removed. Repeating this operation multiple times allows to stretch the image down while preserving most of the interesting content and reduces deformations.

Figure 1: Left to right: (a) Original image. (b) Stretched down: the image is deformed! (c) Cropped: we lost the kid! (d) *Seam Carving*.

Description of the method

This description, without lack of generality, assumes that we are scaling down the image horizontally, by computing a vertical seam.

The **first step** is to determine what kind of features need to be preserved, ie., what would not go unnoticed if removed. A simple heuristic is that we want to preserve edges: any flat uniform surface can be stretched down without producing much artifacts, but stretching down a tree may pose more problems. The first step is thus to compute an *energy map* that detects edges. A simple approach for that is to consider that the value of the energy map $E(x, y)$ at the pixel (x, y) is given by

$$E(x, y) = |I(x + 1, y) - I(x, y)| + |I(x, y + 1) - I(x, y)|$$

where $I(x, y)$ is the intensity of the original image. For color images, a simple approach is to take

$$I(x, y) = R(x, y) + G(x, y) + B(x, y)$$

where R, G, and B are respectively the red, green and blue channels of the image.

The **second step** is to compute a cumulated energy map, $C(x, y)$, that describes optimal paths. Such methods come from the field of dynamic programming, but are relatively easy

Figure 2: Original photo and the associated energy.

to understand. This map can be computed with a rather simple formula:

$$C(x, y) = \min(C(x-1, y-1), C(x, y-1), C(x+1, y-1)) + E(x, y)$$

where E is the energy map above. This function tells you that if you currently are at the position (x, y) in the image, on row y, your best move to go to row $y-1$ is to either go toward the pixel $(x-1, y-1)$ or $(x, y-1)$ or $(x+1, y-1)$ (choosing the one that minimizes this cumulative energy map).

Figure 3: Energy and the cumulative energy (here, $y = 0$ is the top row).

The **third step** is to go to the very last row of the cumulative energy map, and select the pixel with the lowest cumulative energy. This is the starting point of the seam.

The **fourth step** is to start from the pixel chosen in step 3, and progressively build the seam. For that, you will use the insight given in step two: if your current pixel is (x, y), your best move to build the seam is to choose the pixel that minimizes the cumulative energy $C(x, y)$ among pixels $(x-1, y-1)$, $(x, y-1)$, and $(x+1, y-1)$. This operation is called *backtracking*.

The **final step** is to remove the seam. A simple way to see that is that all pixels on the left of the seam remain unchanged, while all pixels on the right of the seam are translated by one pixel to the left. After this step is done, you can finally crop your image by one pixel: you now have successfully rescaled the original image width by one pixel. Repeat the operation as necessary to reduce the image width by the desired amount.

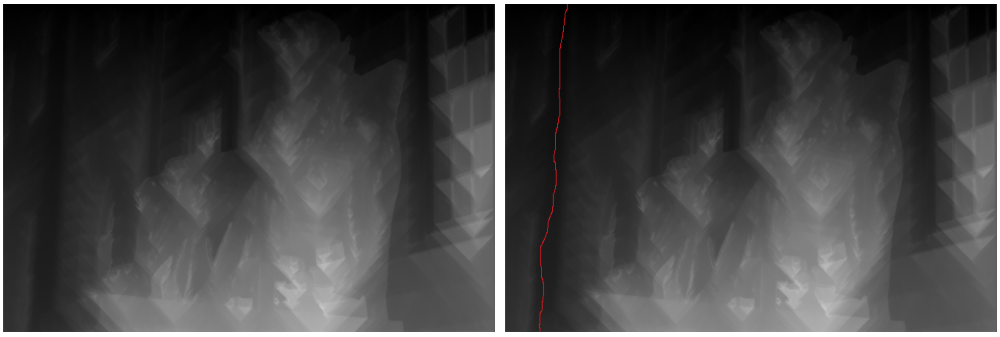


Figure 4: Cumulative energy and a seam that goes through consecutive pixels of lowest cumulative energy (note: this particular seam is not accurate and was done with Paint!).

P1A: Implementation on the GPU

Your first task is to implement Seam Carving on an image to scale the image down horizontally on the GPU.

To do that:

- Compute the **first step** (the value of $E(x, y)$ for all pixels) using appropriate 2D blocks. This step is embarrassingly parallel and is similar in spirit to the sharpening CUDA kernel from HW4.

- The **second step** is more tricky to parallelize on the GPU. A first thing to note is that each row needs to have all the previous rows computed: this requires a global synchronization of all threads, which is only achieved when a given kernel ends. As such, your kernel will only take care of a single row, and will be called as many times as the height of your image.

- For the **third step**, although parallel methods to find the minimum value of a row do exist, you are not asked to implement it in parallel: you should implement a kernel that will be called once with a single thread of the GPU. Please, comment in your writeup why it would not be appropriate to perform this step on the CPU even though a single thread is used (and the rest of the pipeline is on the GPU).

- Implement the **fourth step** similarly on the GPU using a kernel that will be called using a single thread.

- Finally, implement the **final step** using appropriate 2D blocks.

Allow for an arbitrary horizontal rescaling by repeating this procedure. Minimize the amount of communications and retrieve the result from the GPU to the CPU at the end of the computation to save the final image. Do not transfer more data than necessary: the entire pipeline should run on the GPU.

P1B: MPI+CUDA

An interesting application to such methods is the retargeting of video clips from one device (eg., a home cinema) to another (eg., a smartphone).

We will consider that this application is embarrassingly parallel across frames and parallelize it with MPI. You are provided with a sample code that reads a video file, converts it to grayscale and writes an output file. This sample processing is done per-frame in serial, and per-pixel using CUDA; you will need to parallelize across frames as well. You are also provided with a Harry Potter sample video to test this retargeting.

Extra-Credit: Horizontal seams and stretching up

Part I

Your current implementation is limited to stretching down videos horizontally. Generalize your application to horizontal seams, and also allow for scaling up. While vertical scaling is a trivial generalization of horizontal scaling, stretching up is actually not so difficult either. A simple method to scale up is to determine the optimal seam in exactly the same way, but only replace the final step to translate the pixels at the right of the seam to the right (for an horizontal stretch), and fill the gap by repeating adjacent pixel values.

Part II

Using variables in a CUDA kernel can be costly: for instance, multiplying 5 variables is more costly than multiplying 5 hard coded constants (in the latter case, this is done once when the kernel is compiled). As such, it can sometimes be interesting to generate CUDA code dynamically, with, for example, the appropriate hard-coded constants.

Your task for the second part of the extra credit is to use the templating engine *Cheetah* to avoid the need for passing to your kernels the size of your image and any other parameters that can be known at compile time.

Resources

Cheetah engine
Original paper: Seam Carving for Content-Aware Image Resizing

Submission Requirements

- `P1.pdf`: Explanation and analysis of your results, and answers to the specific questions.
- `P1A.py`: Seam carving on the GPU for images (horizontal stretch down)
- `P1Aout.png`: The output of your algorithm applied to the provided sample input image (testimage.jpg) to stretch the image down by 30%
- `P1B.py`: MPI+CUDA video seam carving (horizontal stretch down)
- `P1EC.py`: (optional) MPI+CUDA seam carving generalizations (horizontal+vertical stretch down+up) and meta-programming with Cheetah.