Graphcut Textures: Graph Cuts based Image Synthesis Acceleration

Mingxiao An, Wenxuan Ou, Xiaoyan Wu

Carnegie Mellon University

Problem

Accelerate the Graph Cut based Image Synthesis Algorithm

- Image synthesis is the process of creating new images from some form of image samples.

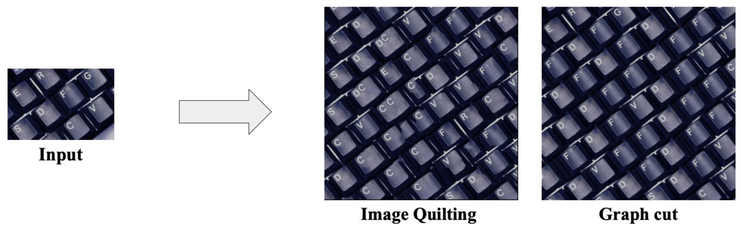

Fig 1. Example output of Graph Cut

- First we pick rectangular candidate patches.
  - Random Placement: Entire input image is translated to a random offset location.
  - Entire Patch Matching (EPM) with FFT-based Acceleration:
    - Search for translations of the image that match well with the currently synthesized output.
    - EPM could produce higher quality output images than random placement.
- Then the Graphic Cut algorithm pick the optimal portion from the patch.
  - Searched for a suitable location to place the patch by the probability produced by EPM.
  - Used the graph cut technique to find the best area, i.e. the minimum graph-cut, for the patch to be transmitted to the output.

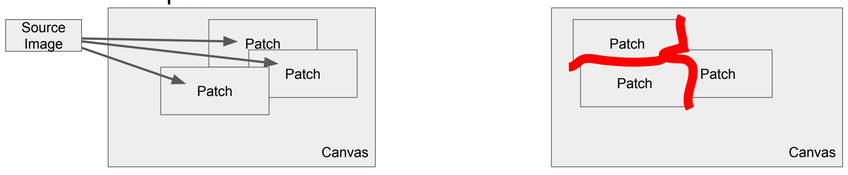

Fig 2. Processing stages of Graph Cut

Comparisons and Analysis Results

We evaluate the efficiency performance by a benchmark

- The benchmark consists of 6 images of various size.
- The original solution has potential memory leak. The baseline is expected to finish producing a 512x512 image in around 850s, but it would encounter out-of-memory error even on a 16GB instance.

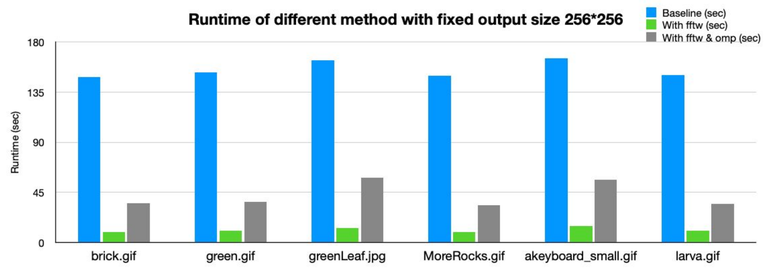

Fig 3.Runtime comparison for each method with different input images

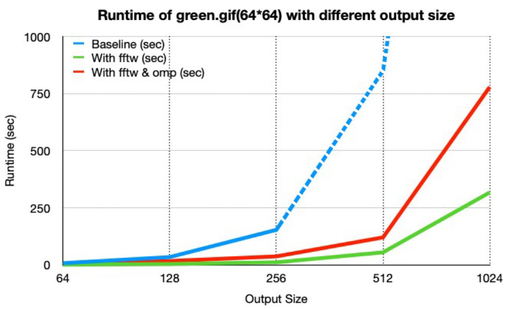

Fig 4. Runtime plot for each method with different output size (Dotted line represents estimated values)

We conducted two comparison experiments

- In the first experiment, we evaluate the runtime of each method on different input images. The output size in this experiment is set to a fixed value 256x256.
  - As shown in Fig 3, the baseline would produce the output in around 150s. Using FFTW library, the runtime is reduced to around 10s, which is impressing. However, using FFTW together with OpenMP features enabled makes the performance dropped to 35-60s with significant variance.
- In the second experiment, we compared implementations on the green.gif image with various output size.
  - As shown in Fig 4, we can still conclude that using FFTW can significant boost the performance compared to the baseline.
- It is interesting that using OpenMP may not be the best option for the current problem scale. One potential reason is the overhead of managing threads. Also, it may also suffer from false sharing from FFTW internal implementation that not manageable by the user.

Methods

Bottleneck Focusing Approach: optimize DFT in Entire Patch Matching

- *Why DFT is the core process to optimize?*
  - The DFT is executed to compute the probability of where should the patch applies, which is frequently invoked every times before applying a refine patch.
  - We profiled the time cost for each part of the procedure, and we found out that the major bottleneck of the baseline is the customized sequential DFT function implemented by the original author.
- *How to boost the efficiency of DFT ?*
  - To boost the performance of DFT and IDFT computation, we used FFTW performance library with the configurations enabling SSE2, AVX, AVX2, and AVX512 instruction sets, as well as the OpenMP for manycore processing.
  - We also apply OpenMP to the loop of canvas-level operations, e.g. load image into DFT domain, DFT multiplication, and possibility computation.

[ref] : https://www.cc.gatech.edu/~turk/my_papers/graph_cuts.pdf

Discussion and Conclusions

FFTW is efficient. Use it instead of building wheels on your own

- *Execution environment*
  - When we were testing the benchmark, we experienced memory overflow for large output size.
  - The compiling of the original framework requires cmake>3.17, so we provisioned a c5.2xlarge from AWS to run the experiments.
- *Performance Conclusions*
  - Using FFTW library can significantly boost the efficiency of DFT and iDFT transformation processes.
  - Because the DFT and iDFT phase contribute to most of the running time of the algorithm, we can observe a significant overall improvement on computation efficiency.
  - Using manycore libraries, i.e. OpenMP, does not further improve the performance at the current scale. The most likely reason is the insufficient scale of the problem. We expect the OpenMP parallel would come into effect with larger output image size. Unfortunately, we cannot afford to produce output images of larger size.

[code]: https://github.com/18646Group/GraphCut