Parallelized String Art

Catherine Tianhong Yu: tianhony@andrew.cmu.edu
Nanxi Li: nanxil1@andrew.cmu.edu

1 URL

https://nanxili.github.io/15418-threadart/

2 Summary

We aim to parallelize the the problem of converting an input image to string art instructions on NVIDIA GPUs using CUDA.

3 Background

First of all, what is string art? String art is a technique for the creation of visual artwork where images emerge from a set of strings that are spanned between pins[1]. Here is a video explaining how it is done: https://vimeo.com/175653201.

We are going to use the algorithm presented by Birsak et al.[1]. This algorithm will not produce a solution using a single continuous thread, but many discrete edges.

Our algorithm computes a subset of all possible strings which are used together to reassemble the input image. In each iter ation we pick the edge that allows the biggest norm reduction, and we stop when further addition would cause an increase of the error.The big disadvantage of a greedy approach as we use it is that a decision at an early stage can later turn out to be a bad choice. In our case, an addition of one edge might bring the biggest benefit when it is chosen, but can then prevent a better solution later on. Thus, we try to further improve the results by an iterative removal of edges. In particular, if the initial addition stage terminates, we sequentially remove those edges that allow to improve the norm. If it is not possible anymore, we start to add edges again, pick one edge at a time, and continue as long as the norm can be improved. We alternate those two stages until it is not possible to further im- prove the norm, neither by removal nor by addition, in which case the algorithm terminates. (Birsak et al, 2018)

With N needles, there are $N \times (N-1)$ edges total(each needle can be connected with all $(N-1) needles$), so each iteration of the while loop runs in N^2 to find the argmin. This $O(N^2)$ computation can be parallelized and finding L2-norm for each edge is independent from one another.

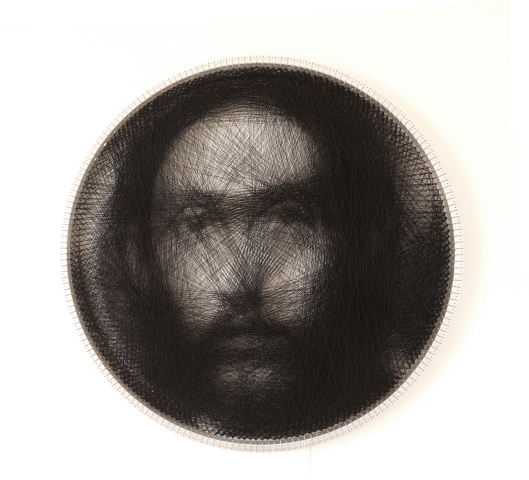

Figure 1: a piece of string art by petros vrellis[4]

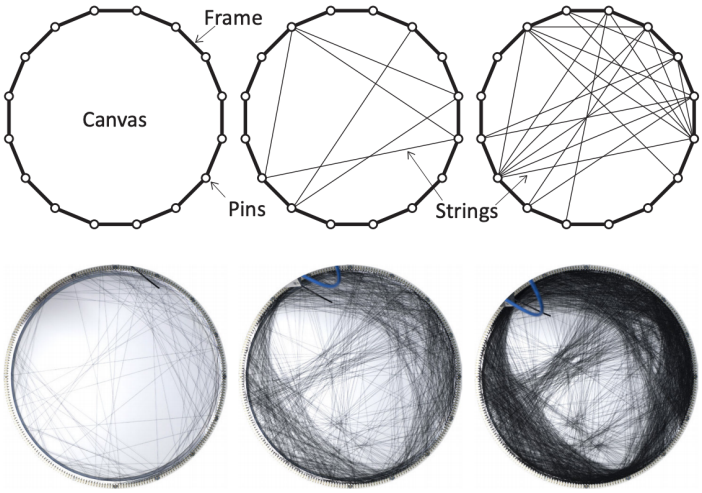

Figure 2: how string art is constructed[1]

In the adding/removing steps, the order of adding and removing here is fixed in the sequential version as it always adds/removes one that minimizes the L2 norm. Potentially in our parallelized version, we can have a less greedy algorithm to take advantage of the parallelism.

The paper also explains how to use auxiliary edges to transform the solution to a fabricable string art image. But for our project, we initially will not implement this part and only output a solution of edges, and if time allows, we will implement adding auxiliary edges.

while *true* **do**
  $j = \arg\min_i \|WF(x \pm e_i) - Wy\|^2$
  $\tilde{f} = \left\|WF\left(x \pm e_j\right) - Wy\right\|^2$
  if $\tilde{f} < f$ **then**
    $x = x \pm e_j$
    $f = \tilde{f}$
  else
    break
  end
end

Algorithm 1: Our algorithm computes a subset of all possible strings which are used together to reassemble the input image. Please note that this code summarizes the edge addition $(x + e_i)$ and edge removal $(x - e_i)$ operations of the algorithm as described in Section 5. Note that the vector e_i refers to the i-th column of the identity matrix.

4 The Challenge

The first challenge of this project is implementing the algorithm itself in C/C++. In addition to the psuedocode presented in the paper, there is are matlab[2] and Java[3] implementations. We need to first translate the implementation into C/C++.

To parallelize finding the minimum of L2-norm for all edges, each thread needs to have access to the minimum. For CUDA, there will be a lot of latency to access global minimum. With the greedy algorithm, a barrier is always needed to find `argmin`. Another challenge would be thinking about how to divide the work to take advantage of parallelism to reduce these sequential work.

5 Resources

We have 2 implementations of the algorithm to look at[2, 3], and the paper[1] itself, but we will be implementing it in C/C++ from scratch. We still need to investigate to find the libraries that can be useful.

We also have access to documentations of CUDA.

As for machines, we will use the GHC machines for CUDA implementation.

6 Goals and Deliverable

PLAN TO ACHIEVE:
We plan to implement the edges solution(instead of a continuous thread) sequentially, in parallel using CUDA.
We will benchmark their performances on GHC and lateday machines and analyze them.
For the demo, we want to be able to take input images, and output the string art version of it using our

parallel algorithms, and show precomputed speedup graphs.

HOPE TO ACHIEVE:
We hope to implement the fabricable solution(using a continoous thread), but even if we do, we highly doubt that we would have the time to make this additional part parallelized, so it would not really contribute to the speedup.

BARE MINIMUM:
If our progress is slower as planned, we will at least implement a parallel version using CUDA without much optimization, but all the analysis and demo will be the same: without Cuda.

7 Platform Choice

We chose CUDA because the same set of instructions are applied to many different edges, and the problem itself lives in the world of computer graphics, so we are curious to see how CUDA could improve performance.

8 Schedule

November 2nd: Discuss Idea with Instructors Monday
November 4th: Project Proposal Due Wednesday
November 5th-9th(5days): Research and find useful libraries to use in C and C++
November 10th-18th(9days): Implement the sequential solution
November 19th-23th(5days): Implement a baseline CUDA version
November 25-December 3rd(10days): Add optimizations to the CUDA version
November 27th: Checkpoint Report
December 3rd-8th(6days): Benchmark CUDA
December 8th-13th(6days): Write report
TBD(after December 13th, day before final exam slot): Final Report Due
TBD(after December 14th, during final exam slot): Virtual Poster Session

9 References

References

[1] Michael Birsak et al. “String art: towards computational fabrication of string images”. In: *Computer Graphics Forum*. Vol. 37. 2. Wiley Online Library. 2018, pp. 263–274.

[2] Exception1984. *Exception1984/StringArt*. URL: https://github.com/Exception1984/StringArt.

[3] Jblezoray. *jblezoray/stringart*. URL: https://github.com/jblezoray/stringart.

[4] petros vrellis petros. *A new way to knit (2016)*. URL: http://artof01.com/vrellis/works/knit.html.