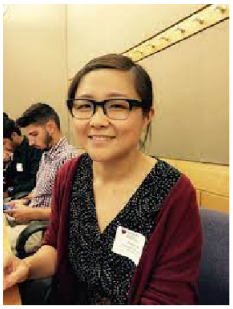
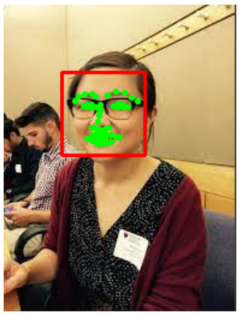
This project seeks to parallelize a shared memory facial recognition algorithm. We discovered a Github repository called **FaceX** (`https://github.com/delphifirst/FaceX`), which performs face contour recognition as demonstrated in the images below.

There are two main components for the application: **FaceX-Train** and **FaceX**, the former trains face regressor model based on labelled sample images for the latter to perform real-time face recognition – the green dots for face features and the red square for face area. We aim to optimize both components.

The documentation for this repository can be found here





Why this Project is Important

We consider this project important because facial recognition programs are used in a wise range of applications including security, personalization, and accessibility. It plays a crucial role in preventing fraud, identifying suspects in criminal investigations, users authentication, and much more. However, while face recognition algorithms offer incredible potential benefits, they also come with great challenges, two of which are speed and scalability. When face recognition algorithms need to perform in real-time scenarios, they are expected to work fast to be effective. Also, they should be capable of scaling up to large datasets to accurately recognize a diverse range of faces with different features, colors, ethnicities and genders. Hence, our goal is to optimize the sequential codes of **FaceX** and **FaceX-Train** to achieve higher efficiency and scalability through parallelization.

Parallelism Offered by the Project

First, we plan to optimize vector operations in the **FaceX-Train** component, especially with regards to the allocation and deallocation of the memory pointed to by the vectors. The repeated allocation and deallocation takes up a considerable amount of time (such as with push_back). When we perform acceleration later on, we will pre-allocate a block of memory for these vector operations to use. Similarly, allocating contiguous memory space helps to increase the cache hit ratio. Second, for image algorithms, float precision is usually sufficient, so in order to reduce memory usage and increase speed, we will change the double data type used in this program to use the float data type for training.

In optimizing and accelerating this program, some internal functions of OpenCV have already used SIMD and multi-threading. For these functions, we will look for possible operation mergers from the algorithm level. At the same time, through performance analysis, we found that there are still a lot of code in **FaceX-Train** that is suitable for OpenMP and SIMD parallelism, such as **ShapeDifference**, **ShapeAdjustment**, **Covariance**, and related loops. They can be parallelized with OpenMP and SIMD. Our project will experiment with different means of parallelism with OpenMP and MPI. We will analyse the performance increase achieved from using each of these individually and layering one on top of another.

We will assess performance using the roofline model, and in doing so, measure the reduction in memory bound and computation bound resulting from the optimization of our library.

Memory Models Used

We plan to use SIMD since faceX does a lot of matrix calculation where it applies the same function to each row of a given matrix obtained from opencv image stream. The rows are independent of one another which implies a prime source for data level parallelism and SIMD model.

Potential Challenges

First, one speedup we would implement is parallelizing operation by row. We find that the alignment function would take in 2 rows at any iteration and the rows are independent of each other. As such we can divide up the matrix into copies of 2 rows for each thread to process. One potential challenge would be that the cache for each thread may not be able to hold a full row of data and they would compete to search down the memory pyramid, resulting in further slowdown compared to sequential code.

Second, since OpenMP and SIMD are only limited to the same node, as the size of training samples increases, the training time will increase. The computing resources of a single node are no longer sufficient. Therefore, we will consider using MPI to run multiple nodes in parallel for training. However, it is necessary to consider the trade-off between communication and calculation when dealing with large amounts of data, as communication between different nodes through MPI will also take time.

Current Performance Assessments

We run both the **FaceX** and **FaceX-Train** on a PC with 6 core CPU, 32GB memory and GTX 1070 graphic card. For the former component, it ran with an average lagging time of 1.5 seconds between face update and face recognition update; For the latter component on 560 250 $\times$ 250 pixel images, it takes 1 hour and 15 minutes for 10 training iterations to run.

We believe that there are plenty of rooms for optimization where the upper bound of computing resource requirement would be reasonable. We may even be able to run the optimized version of **FaceX** on our own PC entirely.