DIP Final Report

Team Members

- R07944007 林良翰 - Meshflow implementation and improvement
- R08942124 吳佳展 - Experiment (motion propagation)
- B03902099 黃奕杰 - Report, slides
- D06922010 林昶丞 - Experiment (mesh warping)

Requirements

- Python 3.7
  - Packages - matplotlib, opencv-python, numpy, scipy, tqdm, cvxpy
  - Install requirements

    ```
    pip3 install -r requirements.txt
    ```

- Data
  - Our sample video are stored in google drive, you can download it from here.

Program Usage

- Stabilize a video

  ```
  python3 -m src.stabilization [source_video_path]
  [output_video_path]
  ```

- Too see more parameters and options

  ```
  python3 -m src.stabilization --help
  ```

Works & Experiments

We implements MeshFlow as our stabilization method, and improves its speed while maintaining the performance. The following are 2 main sections about our code works. The first part is bottleneck reseaching, which helps us to find out the slowest part of the original code. The second part is our code works, which improve motion propagation and mesh

warping before and after trajectory optimization. To reproduce our experiment results, checkout our notebook.

Bottleneck

We write a simple timer utility to test each part of the original code, and take small-shaky-5.avi (210 frames) as our experiment example input video.

- Construct vertex profiles

	Read Frame	Feature Points	Optical Flow	Motion Propagation	Expand Dimension	Vertex Profiles	Total
Time / Frame (ms)	0.19	0.86	0.19	92.62	0.1	0.1	**94.05**
Total Time (s)	0.04	0.18	0.04	19.45	0.02	0.02	**19.75**

  The motion propagation function uses deep nested for loops (3 layers), and has lots of re-computations of Euclidean distance between vertices and mesh vertex positions.

- Optimize camera path $\approx$ 2 s
- Output stabilized frames

	Read Frame	Mesh Warping	Resize	Write Frame	Total
Time / Frame (ms)	0.14	149.62	0.19	1.1	**151**
Total Time (s)	0.03	31.42	0.04	0.23	**31.71**

  In mesh warping function, the program still uses deep nested for loops (4 layers). The mesh vertex positions and vertex transformations are calculated in side for loops, which causes a huge bottleneck.

- Total time elapsed $\approx$ **55.24** s

Improvements

The following tables show that our code really improves the speed of meshflow.

- Construct vertex profiles

	Read Frame	Feature Points	Optical Flow	Motion Propagation	Expand Dimension	Vertex Profiles	Total
Time / Frame (ms)	0.19	0.76	0.19	5.95	0.05	0.05	**7.19**
Total Time (s)	0.04	0.16	0.04	1.25	0.01	0.01	**1.51**

Instead of using Euclidean distance for distributing feature motion vectors, we use L1 distance for much lower computation cost while maintaining the final stabilization performance. Also, we reorder the for loop layers to prevent re-computations.

- Optimize camera path ≈ 2 s
- Output stabilized frames

	Read Frame	Mesh Warping	Resize	Write Frame	Total
Time / Frame (ms)	0.14	10.14	0.19	1.00	**11.52**
Total Time (s)	0.03	2.13	0.04	0.21	**2.42**

We remove the inner 2 for loop layers for computing the pixel position inside grid meshes and implements numpy `meshgrid` function to speed up the computation time and avoid re-computations. Besides, we calculate the transformation of pixels with numpy `dot` operation to process pixels in batch.

- Total time elapsed ≈ **7.63** s

Result

The following table is the final comparison between original and improved code with multiple videos. Our improvement makes meshflow **10.38**x faster than original code.

Video	parallax	running	sample	selfie	simple	shaky-5	Total
Original (s)	890.62	684.08	428.36	371.21	434.18	690.14	3498.59
Improved (s)	71.90	60.35	46.30	49.31	48.20	60.96	337.02
Speed Up Rate	12.39x	11.34x	9.25x	7.53x	9.01x	11.32x	**10.38**x

Result download link - https://drive.google.com/open?id=1o5Ybedo7gt4w-jFcA9uzd0UsRVVgwOrS