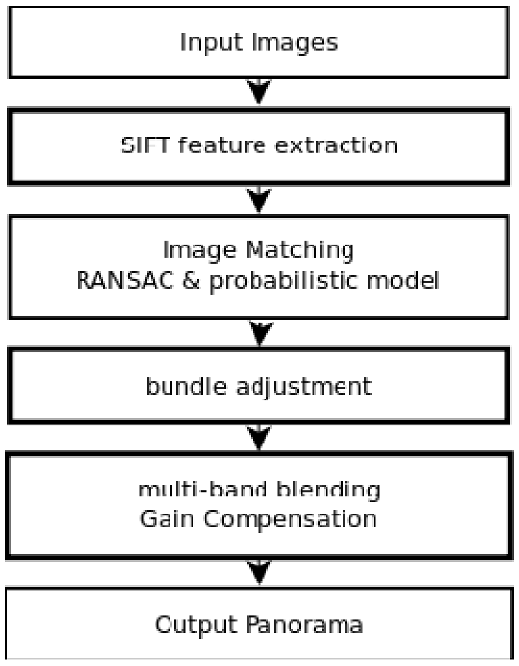
Automatic Panoramic Image Stitching Using Invariant Features

B95902049 陳耀男 B95902106 溫在宇 B97902115 郭冠宏
R98922037 郭彥伶 R98922069 詹承偉 R98922165 蔡青樺

System Diagram

```
Algorithm: Automatic Panorama Stitching
Input: n unordered images
  1. Extract SIFT features from all n images
  2. Find k nearest-neighbors for each feature using a k-d tree
  3. For each image:
       I.   Select m candidate matching images that have the most feature
            matches to this image
       II.  Find geometrically consistent feature matches We using RANSAC to
            solve for the homography between pairs of images
       III. Verify image matches using a probabilistic model
  4. Find connected components of image matches
  5. For each connected component:
       I.   Perform bundle adjustment to solve for the rotation and focal length $f$
            of all cameras
       II.  Render panorama using multi-band blending
Output: Panoramic image(s)
```



SIFT

I. **Scale-space extrema detection**

This is the most time consuming stage. In this stage the input image will apply many Gaussian filters with different variance. Because each pixel is independent from each other we can compute then parallelly.

For each block we will have 16x16 threads. So the total number of blocks will be $\lceil w/16 \rceil$ x $\lceil h/16 \rceil$ where w and h is the width and height of the input image respectively. We can guarantee that global memory access is fully coalesced.

To further improve the performance we use a 2D texture memory to store the input data. This improves the memory throughput because threads in the same blocks will access nearby memory location in a 2D-space.

We also use share memory to store Gaussian filter table because each thread use the same set of table.

II. **Discarding low-contrast key points & Eliminating edge responses**

After finding the feature location we eliminate key point which will produce unstable feature.

III. **Orientation assignment**

In this step, our task is computing these feature points' orientation. First, take the gradient of points which distance is less than sigma and do weighted sum with Gaussian. Now, we have a histogram of angle. We slightly smooth the angle histogram. Finally we take the maximum angle and other angle that bigger than 80% of maximum as our feature orientation. In parallel, for each thread, compute one feature point.

IV. **Key point descriptor**

For each key point and orientation pair we compute there SIFT descriptor using local image gradient magnitude and angle. Each key point will collect data from a 128x128 pixel square and store the result in a 128-dims vector. The image is store in global memory so is the vector. All the operation requires a lot of memory bandwidth result in low instruction throughput. Our solution is to break the computation of the 128-dims vector in many different threads. Each thread will compute 8 entries of the 128-dims vector. So we can store the vector data in 8 registers instead of global memory. Although we need to compute the same information in different thread but the overhead is small compare to the gain of reduce memory access.

Sequential Version

Function	Time (sec)	Percentage
Gauss	7.84	82.68%
DOG + MOVE	0.32	3.38%
Gradient	> 0.01	0%
Orientation	0.08	0.84%
Descriptor	1.23	12.99%
Total	9.47	

First CUDA Version

Function	Time (msec)	Percentage
Gauss	59404.5	14.80%
DOG + MOVE	29356.24	7%
Gradient	7417.18	2%
Orientation	12277.4	3%
Descriptor	225552.67	56%
Misc	735.968	0.20%
Memcpy	65390.8	16%
TOTAL	400134.758	

Final CUDA Version

Function	Time	Percentage
Gauss	29197	19%
DOG + MOVE	19648.108	13%
Gradient	4550.308	3%
Orientation	11770.2	8%
Descriptor	66671.97	45%
Misc	736.192	0.50%
Memcpy	16724.3	11%
TOTAL	149298.078	

The above table shows the time used for each step in sequential version and CUDA version. As you can see we gain most of the speedup from Gaussian Filter as the CUDA version is 270 times faster than the sequential version. When optimize the CUDA version we try to reduce the must time consuming step "**Descriptor**". Using method like reorder memory access and re-compute most of the data instead of storing them. We are able to speedup this stage about 3.4 times faster.

The overall speedup is about 67 times faster than the sequential version.

Feature Matching

For each feature, find 4 nearest feature (128 dimension and according to L2 loss) for matching which is the next step.
For each thread in CUDA, it takes its own feature and goes through all other features to find 4 nearest features.

Parallel Optimization Detail

1. **Shared Memory**
   Let every thread load its own feature to shared memory and be expected that It will be faster. But the result seems to slower than using global memory.
2. **Exchange Dimension**
   Because the instruction executing in all threads in CUDA is the same; therefore, when it take feature for computing L2 loss, it is better to let the location of memory be closer. Therefore we change dimension from [Total Feature][128 dimension] to [128 dimension][Total Feature]. After exchange, when all thread load first dimension the locality is better than original.
3. **Register**
   First, we use the array for storing the nearest feature and distance as far. However, array will be stored in global memory. Therefore, we change this array to 4 batch variable. Variable will be stored in register. Therefore, it will speed up.
4. **Texture**
   Because there is cache in texture and our feature locality, it will be better if we using texture. After using texture, it slightly speeds up. But we have a problem. If the number of features is very large, texture can't bind the large memory. Therefore, we give up this small speed-up and recover to original version.

Finally the speed-up of feature matching is about 15 times.

Version	Time (sec)
Sequential	17.913
Share Memory	15.221
Global Memory Only	4.97
Exchange Dimension	1.39
Use Register	1.17

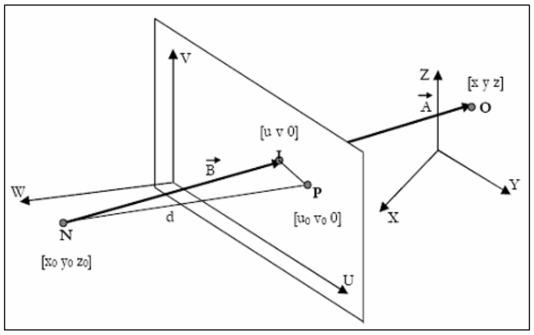
RANSAC

After we have the information of feature matching of all pictures, we can use this useful information to do image matching. In image matching step, we are going to find out which picture is a neighbor of another picture, and find the correctly feature matching set we need for next step of all feature matching set.
First, for each picture we consider 6 images that have the greatest number of feature matches to it, and then we use RANSAC to select a set of inliers that are compatible with a homography between the images.
RANSAC(RANdom SAmple Consensus) is a non-deterministic algorithm estimates parameters of a mathematical model from a set of observed data which contains outliners iteratively. We show the step of RANSAC in details as follow.

RANSAC loop:

1. Select four feature pairs (at random)
2. Compute homography H (exact)
3. Compute *inliers* where $SSD(\, p_i', \boldsymbol{H} p_i \,) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers

We use DLT (Direct Linear Transformation) as the mathematical model in RANSAC. DLT can find the Homography between the images.



DLT 's mathematical model

In the paper we implement run 500 RANSAC iteration to find the model which fits the largest set of inliners, but we figure out that if we just implement as paper says in some times the model RANSAC find is not the correct model matching this two images. In order to reduce the probability of getting wrong model, we run 3000 iteration and find the model has a smallest error sum of inliners.

Image stitching

After the RANSAC stage, we get the homography $\mathbf{H}_{ij}$ between each matched images, e.g. image ***i*** and image ***j***. To stitch the images and remove the distortion, we randomly select an image as a start image ***s*** and project each point in other images, i.e. $\mathbf{x}_j$, to the coordinate of the start image ***s*** by $\mathbf{x}_s = \mathbf{H}_{sj}\mathbf{x}_j$.

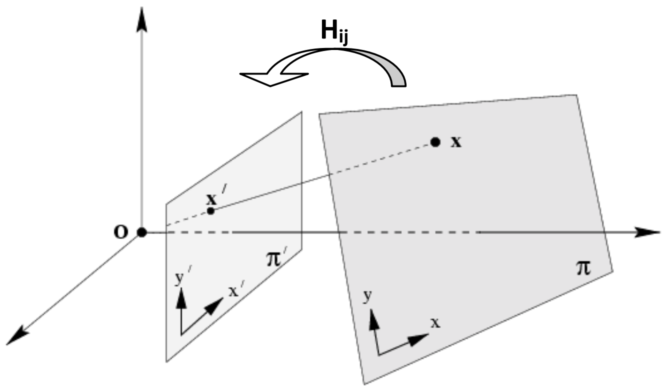


Figure 1 Image transformation using homography $\mathbf{H}_{ij}$

For each neighbor image ***j*** of ***s***, we use $\mathbf{x}_s = \mathbf{H}_{sj}\mathbf{x}_j$ to calculate the new position of the points of ***j***. For other images i, we need to find a sequence of linked images and transform the image by a series of homography matrices, i.e. $\mathbf{x}_s = \mathbf{H}_{sm}\mathbf{H}_{mn}...\mathbf{H}_{hi}\mathbf{x}_i$. The stitched image is the transformation result of each image as shown in Figure 2.

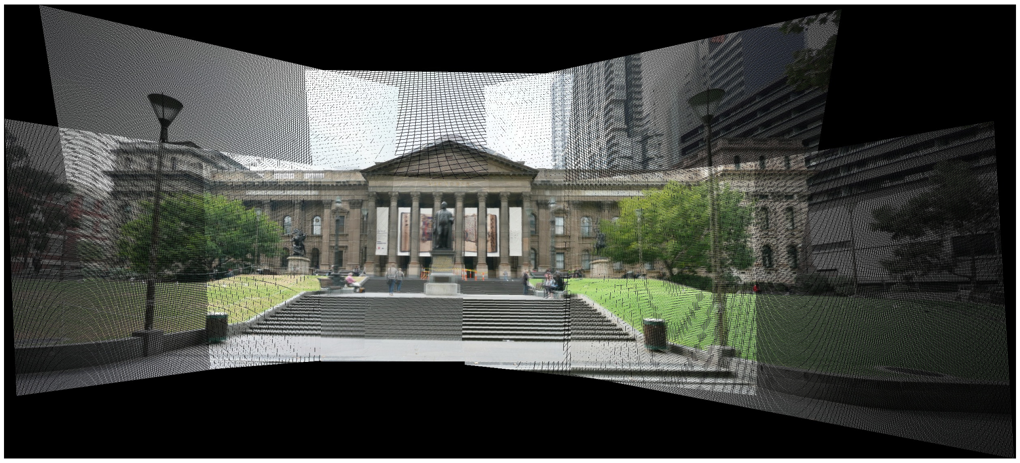
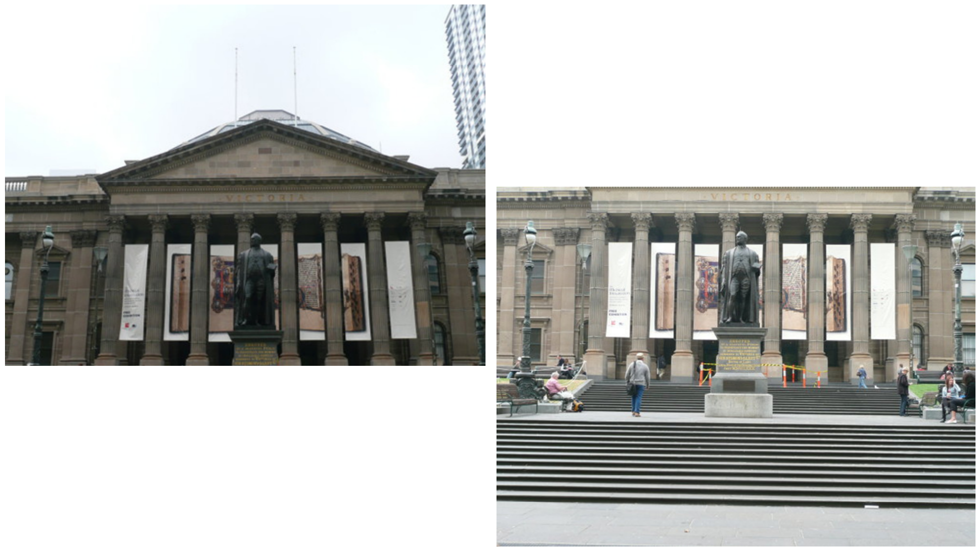


Figure. The stitched image after RANSAC stage

Gain Compensation

Since there are obvious differences in brightness between overlapped images, finding overall gains to adjust brightness for images becomes necessary.



1. Start from setting an error measurement function:

$$e = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{\substack{\mathbf{u}_i \epsilon \mathcal{R}(i,j) \\ \tilde{\mathbf{u}}_i = \mathbf{H}_{ij} \tilde{\mathbf{u}}_j}} (g_i I_i(\mathbf{u}_i) - g_j I_j(\mathbf{u}_j))^2$$

   where g indicates the gain vector of images, and $R(i,j)$ is the overlapped region between image i and j.

2. Approximate $I(u_i)$ to the mean of overlapped region $\bar{I}_{ij}$:

$$\bar{I}_{ij} = \frac{\sum_{\mathbf{u}_i \epsilon \mathcal{R}(i,j)} I_i(\mathbf{u}_i)}{\sum_{\mathbf{u}_i \epsilon \mathcal{R}(i,j)} 1}$$

3. Add a term to make the gains close to unity (for avoiding the all zero solution):

$$e = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} N_{ij} \left((g_i \bar{I}_{ij} - g_j \bar{I}_{ji})^2 / \sigma_N^2 + (1 - g_i)^2 / \sigma_g^2 \right)$$

   where $N_{ij} = |R(i,j)|$.

4. Minimize the rewritten error measurement function above by setting the derivative to zero, and get a set of overall gains for image brightness adjustment.

5. Finally, multiply all the pixel values by the overall gains.

Multi-Band Blending

Even a small registration error causes ghost (blurring of high frequency), therefore, multi-band blending is applied when image stitching. That is, blend low frequencies over a large spatial range, and blend high frequencies over a short range.

1. For each output pixel, find the input image which can represent this pixel value best in the stitching:

$$W^i_{max}(\theta,\phi) = \begin{cases} 1 & \text{if } W^i(\theta,\phi) = \arg\max_j W^j(\theta,\phi) \\ 0 & \text{otherwise} \end{cases}$$

$$W(x,y) = w(x)w(y)$$

where $W(x)$ linearly varies from 1 at the center to 0 at the edge of input image.

2. Render high frequency part in the images:

$$\begin{aligned} B^i_\sigma(\theta,\phi) &= I^i(\theta,\phi) - I^i_\sigma(\theta,\phi) \\ I^i_\sigma(\theta,\phi) &= I^i(\theta,\phi) * g_\sigma(\theta,\phi) \end{aligned}$$

Blur the max-weight functions to form the blending weights:

$$W^i_\sigma(\theta,\phi) = W^i_{max}(\theta,\phi) * g_\sigma(\theta,\phi)$$

3. Calculate for lower frequency bands:

$$\begin{aligned} B^i_{(k+1)\sigma} &= I^i_{k\sigma} - I^i_{(k+1)\sigma} \\ I^i_{(k+1)\sigma} &= I^i_{k\sigma} * g_{\sigma'} \\ W^i_{(k+1)\sigma} &= W^i_{k\sigma} * g_{\sigma'} \end{aligned}$$

where $\sigma' = \sqrt{(2k+1)}\sigma$.

4. Finally, combine the values from each band as the final result:

$$I_{k\sigma}^{multi}(\theta,\phi) = \frac{\sum_{i=1}^{n} B_{k\sigma}^{i}(\theta,\phi) W_{k\sigma}^{i}(\theta,\phi)}{\sum_{i=1}^{n} W_{k\sigma}^{i}(\theta,\phi)}.$$

Parallelize the Gaussian convolution part, and optimize it by binding the Gaussian matrix with texture or using shared memory to store the Gaussian matrix. The below table shows the execution times of sequential version, parallelized version, optimized with texture version and optimized with shared memory version. The optimized version speeds up 5.54X over the sequential version.

Version	Time (sec)
Sequential	36.33
Parallel	7.20
With Texture	6.62
With shared memory	6.55

Total Time Comparison

Sequential Version

Function	Time	Percentage
SIFT	53.95	7.42%
Match	618.80	85.08%
RANSAC	21.57	2.97%
Blend	32.95	4.53%
Total	727	

CUDA Version

Function	Time	Percentage
SIFT (CUDA)	2.53	3.36%
Match (CUDA)	40.09	52.74%
RANSAC	21.06	27.70%
Blend (CUDA)	11.59	16.17%
Total	75.997	

Using CUDA we improve the performance of **Panoramic Image Stitching** nearly 10 times faster. Our system can build the panorama in about a minute when given 6 input images with 1000x800 size where the sequential version takes about 12 minutes to compute the same panorama.

Job Assignments

B95902106 溫在宇	SIFT, Feature Matching, RANSAC
B95902049 陳耀男	SIFT, Feature Matching
B97902115 郭冠宏	Web Interface
R98922037 郭彥伶	RANSAC, Image stitching
R98922069 詹承偉	RANSAC, Image stitching
R98922165 蔡青樺	Gain Compensation, Multi-Band Blending

Reference

M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. IJCV, 74(1):59–73, August 2007.