ABSTRACT

Title of Thesis: A Framework for GPU 3D Model Reconstruction Using Structure-from-Motion

Yu Wang, Master of Science, 2011

Thesis directed by: Marc Olano, Associate Professor
Department of Computer Science and
Electrical Engineering

A framework for three-dimensional (3D) model reconstruction is described in this thesis. The primary application is scanning forest canopies and assisting scientific applications such as fire hazard evaluation and vegetation biomass estimation, using photos taken from a radio-controlled helicopter, although the methods apply to any series of photographs taken along a path. The approach is based on the fact that the photos are taken in a continuous path, thus taking advantage of the adjacency between images. The major contributions of this project are 1) introduce a linear time complexity algorithm that reduces number of image pairs to match for datasets obtained in a continuous path 2) present an optimized high performance framework for GPU 3D reconstruction using structure-from-motion.

A Framework for GPU 3D Model Reconstruction Using Structure-from-Motion

by

YU WANG

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2011

Chapter 1

INTRODUCTION

High spatial resolution measurements of three-dimensional vegetation structure is essential for accurate estimation of vegetation biomass, carbon accounting, forestry and other scientific applications. These applications reconstruct a detailed 3D model from photographs using structure-from-motion(SfM) algorithms and require large collections of photographs. The process to generate the reconstructed geometry is expensive in terms of instrument and labor cost, as well as processing time for the photo data set (Kampe *et al.* 2010). In order to obtain the datasets, researchers have been using aircraft installed with high-resolution cameras and global positioning systems (Dandois & Ellis 2010), which are usually constrained by weather, research budgets, flight schedules and require significant human labor. Reconstruction algorithms match the images in the datasets to find common points and use this information to compute the three-dimensional structure of the object and the pose of the cameras that captured these images.

Researchers at the Geography and Environmental System Department, University of Maryland, Baltimore County, designed a novel method of collecting scanning photographs of local forest canopies for geometric reconstruction. They use radio controlled helicopters with six propellers loaded with a commercial camera and navigate the area of interest, taking photographs in a pre-designed path as in Figure 1.1. Point A is the starting point of

the path, Point B is the ending point of the scanning path and photos are taken continuously as the helicopter flies by the area in path P. The green stars represent the location where a photo is taken by the digital camera. This method, using a low-cost portable platform, effectively reduces research budget and with the help of existing structure-from-motion systems, we can get 3D reconstructions at a relatively low cost.

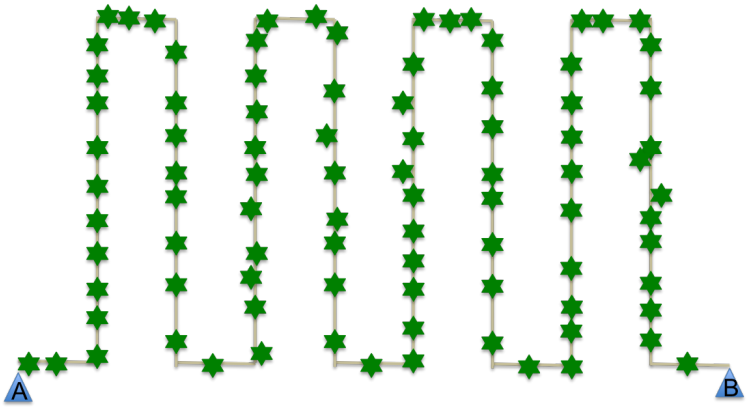


FIG. 1.1. Navigation path of the radio controlled helicopter

The open source software package Bundler (Snavely, Seitz, & Szeliski 2006) is a structure-from-motion system for unordered image collections. It takes in a collection of images, image features and image matches as input, and produces a 3D reconstruction of camera and scene geometry as output. Bundler looks for the common features between every pair of images in the collection and determines whether the two images have sufficient features in common to contribute to the reconstruction of the 3D scene. The time complexity of this pairwise comparison is $O(n^2)$, where n is the number of images in the photo data set because no assumption about the image correlation is made. As a result, the processing time for the Bundler system usually takes days or even weeks to finish the 3D geometric reconstruction from a collection of a few hundred photos.

Under the assumption that the images are taken in a continuous path, we can efficiently reduce the processing time of pairwise comparisons between every two images in the photo set to comparisons between local groups of images that have features in common. As a result, the proposed algorithm reduces the time complexity of the image processing procedure, especially the pairwise image comparisons from every two images in the dataset to only adjacent images.

The reconstruction process is decomposed into several stages (Snavely, Seitz, & Szeliski 2006), which include feature extraction, feature matching, scene reconstruction and camera position approximation. The original Bundler application was fully implemented on the CPU. The ultimate goal would be to implement the framework on the GPU, and this thesis project provides GPU solutions to the feature extraction and feature matching stage of the framework.

By using the graphics processing units (GPU) in combination with NVIDIA's compute unified device architecture (CUDA), I developed parallel algorithms to perform image feature extraction and feature matching in parallel on hundreds of GPU cores. One more stage is added to the reconstruction process in the framework after the geometry is reconstructed: based on the reconstructed camera positions, for each camera, its nearest neighbor is located using a GPU algorithm, so that we get the neighboring images of the image taken by this camera, including ones from adjacent paths. This information is passed back to feature matching stage and the scene is reconstructed using this information.

The high level work flow in the framework is: for each image in the given dataset, image features are extracted using a GPU implementation by Wu (2011) of the SIFT algorithm invented by David Lowe (2004), and the distinctive features in the images are extracted and copied to the CPU; in the first feature matching stage, only adjacent images along the path are matched, and SIFT features of each image are copied to the GPU memory and matched using several CUDA kernels, then results are copied to the CPU; scene reconstruction is

performed using Bundler's bundle adjustment routine, and positions of the cameras are also approximated; in the next stage, k neighbors of each image are located and their information is sent to back to the feature matching and scene reconstruction routine to find correspondences across paths and reduce path-to-path drift in the reconstruction.

The rest of the paper is organized as follows: Chapter 2 covers background and related work, Chapter 3 introduces the proposed framework, Chapter 4 presents the parallel algorithms, Chapter 5 analyze the algorithm complexity and performance, and Chapter 6 concludes.

Chapter 2

BACKGROUND AND RELATED WORK

In this section, some existing applications for point cloud reconstruction and the fundamental theories adopted in these applications are introduced.

2.1 Point Cloud Reconstruction

Agarwal (2009) introduced a framework that efficiently arranges some stages of the reconstruction process among compute nodes and calculates point clouds for famous landmarks from 150K images downloaded from Flickr.com within a day on a cluster with 500 compute nodes. Although the job distribution provides speedup, it is limited by the communication costs. Farenzena et al. (2009) described a partition algorithm that limits the number of features in an image to use for feature matching in the initial approximation, and images are grouped into clusters based on the number of matched features. This method reduces computation cost and the reconstruction drift, but at least 20 active views are suggested to provide reliable and fast computing. PhotoSynth (Microsoft 2010) is an application developed by Microsoft Live Labs and the University of Washington that analyzes unordered photo collections and generates a three-dimensional reconstruction of the photos and a point cloud of a photographed object. Christopher Zach introduced a method (Zach 2011) that reconstructs a 3D model in the form of sparse point clouds.

2.2 Scale-Invariant Feature Transform (SIFT)

A feature is a significant piece of information extracted from an image which provides more detailed understanding of the image. For instance, an edge, a corner, a blob, or a point of interest, etc. Feature detection is to identify the presence of a certain type of feature or object in an image. Algorithms for image feature detection include Canny operator (1986) and Sobel operator for edge detection; Harris operator for corner detection; Laplacian operator for blob detection, etc. For local feature detection and description, David Lowe's SIFT (2004) algorithm is commonly applied by computer vision projects.

David Lowe (2004) proposed a method that applies Gaussian filters of different widths to the input image, so that a sequence of filtered images are generated. By subtracting every two consecutive blurred images, we get a stack of Difference of Gaussian-filtered images (DoG, as shown in Figure 2.1). Local extrema are the invariant features of the image. Wu (2011) implemented (Lowe 2004) using a GPU. The method processes pixels in parallel to build Gaussian pyramids and detect DoG points.

2.3 Structure from Motion(SfM)

Photogrammetry studies modern multi-view geometry techniques, and the main problem is to determine the 3D location of a point in a scene from multiple photographs taken from different vantage points. Using triangulation, any feature seen in at least two photographs taken from known locations can be localized in 3D. The triangulation method can be described in terms of a function F such that

$$x \sim F(y_1', y_2', C1, C2) \tag{2.1}$$

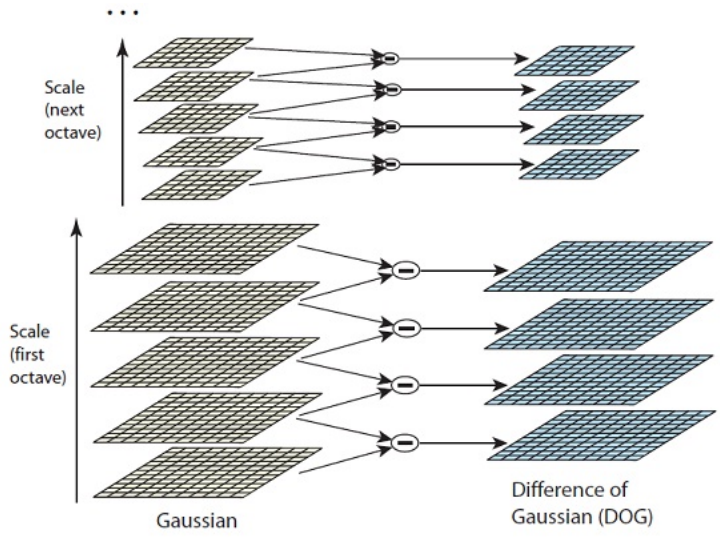


FIG. 2.1. SIFT: difference of Gaussian

where y'_1 and y'_2 are the homogeneous coordinates of the detected image points and $C1$ and $C2$ are the camera matrices, x is the homogeneous representation of the resulting 3D point.

Kruppa (1913) proved that for two views with five-point correspondences, the camera poses and 3D point locations can be determined. Longuet (1981) introduced an algorithm for computing the three-dimensional structure of a scene from a correlated pair of perspective projections, when the spatial relationship between the two projections is unknown.

2.4 Bundle Adjustment

Bundle adjustment (Triggs *et al.* 2000) solves the optimization of 3D coordinates that describe the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera employed to acquire the images. It minimizes the re-projection errors between predicted image points and the image locations which are achieved by using nonlinear least-squares optimization. The problem we need to solve is represented in the

following equation:

$$\min_{x} \frac{1}{2} \| F(x) \|^2 \tag{2.2}$$

Here, x is an n-dimensional vector of variables, representing n 3D feature points; F(x) = $[f_1(x), f_2(x), ..., f_m(x)]^T$, an m-dimensional function of x, representing the internal calibration parameters of m cameras that take an image of the n feature points.

A typical solution to non-linear least squares problem is the Levenberg Marquardt Algorithm (Nocedal & Wright 2000) as shown below:

$$\min_{\triangle x} \frac{1}{2} \| J(x) \triangle x + F(x) \|^2 + \mu \| D(x) \triangle x \|^2 \tag{2.3}$$

Here, $J(x)$ is the Jacobian of $F(x)$, μ is a parameter that controls the strength of regularization, and $D(x)$ is a non-negative diagonal matrix, typically the square root of the diagonal of the matrix $J(x)^T J(x)$. At each iteration, a correction $\triangle x$ is approximated to vector x.

2.5 Faster SfM Methods

A common multi-image situation is that the images are taken in a particular sequence, such as the frames of a video or images captured by a vehicle moving through a building. We can match corresponding points between consecutive images using the basic five-point algorithm to recover the poses of the first two frames and triangulate their common points, then add the next frame, triangulate any new points, and repeat. This method works on short sequences of video clips or photo sets, although the error of pose estimation would accumulate as the size of the data set gets larger.

Another bottom-up solution (Fitzgibbon & Zisserman 1998) is to divide the sequence

of images or video frames into independent sub-sequences, and reconstruct these sub-sequences independently and merge into progressively larger reconstructions.

Some methods based on bundle adjustment are designed for ordered image sequences such as video clips, where some simplifying assumption can be made. One assumption is that a newly added frame only affects several frames preceding this new frame. Previous work has been done based on this simplifying assumption. Cornelis et al. (2006) based on the assumption that the stereo-heads camera internals and relative pose are pre-calibrated, introduced a local optimization method where a fixed-length feature track is reconstructed when the number of images through which it was tracked exceeds a threshold. The length of a feature track is limited to avoid memory congestion during SfM calculations. Pollefeys et al. (2004) based their design on the assumption that a view is only matched with its predecessor in the sequence, and presented a framework that does not require the camera motion nor the camera settings as priori. Although it does not work well when the camera moves back and forth. Once the structure and motion has been determined for the whole sequence, the results can be refined through a projective bundle adjustment.

2.6 K-Nearest Neighbor Search Algorithm

Nearest neighbor search is a type of optimization problem for locating the closest points to query points in metric spaces, formally: Given a set R of reference points in space S and a query point $q \in S$, find the closest point in R to q. An extension of this problem is locating the closest k neighbors of query point q in S based on their Euclidean distances.

A naive solution, also known as the ”brute force” method, to the k-nearest neighbor search is to compute the distance from the query point to every reference point and return the closest k points. This method does not require space complexity besides the reference

points. Garcia et al. (2008) introduced a CUDA implementation of the brute force k-nearest neighbor search algorithm (Figure 2.2) which parallelizes the computation of distances between reference points and query points. In addition, insertion sort is used to collect the first k candidates.

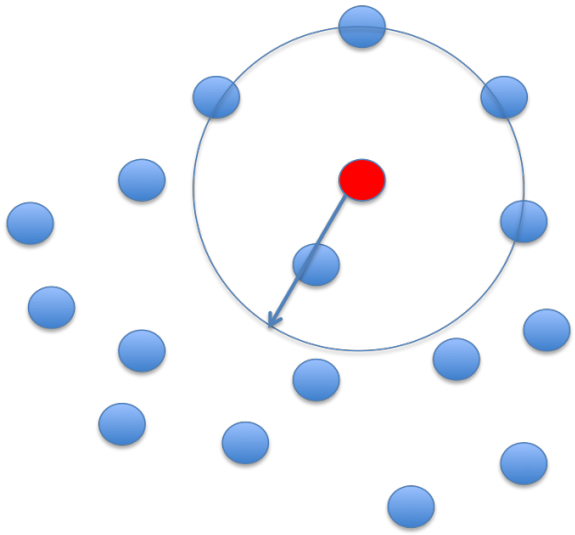


FIG. 2.2. k-nearest neighbors given a query point (k=5)

Local sensitive hashing (LSH) is a method that reduces dimensionality of high-dimensional data by performing probabilistic reduction, and input data are hashed into buckets based on their similarities. Vector approximation files (VA) compress feature vectors and store them in the RAM for a pre-filter on the datasets and the final candidates are determined in a second stage using uncompressed from the disk for distance calculation. Another way to tackle the problem is space partitioning. Arya et al. (1998) proposed an algorithm that builds a k-dimensional tree data structure based on a set of reference points, and given a query point q, the approximate nearest neighbor or k-nearest neighbors are reported. (Figure 2.3). Since there is a restriction on maximum points to consider in each

leave of the tree structure, the kd-tree returns the approximate neighbors.

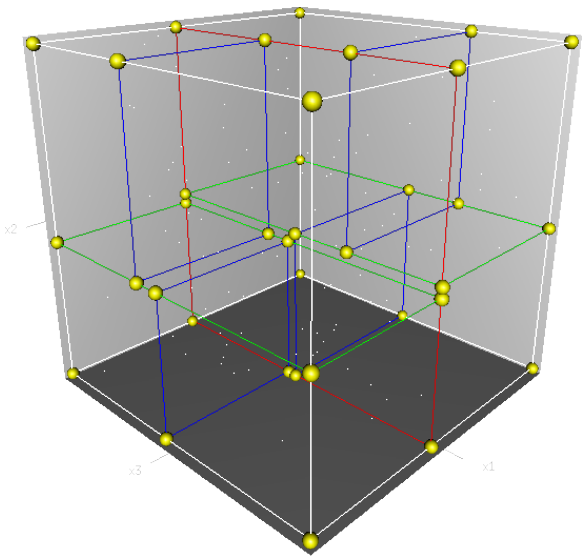


FIG. 2.3. kd-tree data structure, when k=3

According to the research of similarity-search methods by the Bell Lab (Weber, Schek, & Blott 1998), linear scan (brute force) method outperforms methods partitioning space when the dimensionality of data exceeds 10. The feature descriptor of images defined by Lowe's SIFT method is 128 dimensional, which is more than 10, as a result, the theoretical performance of linear method shall be better than tree-structure methods in the feature matching stage. On the other hand, for data with dimensionality of less than 10, the tree-structure methods deliver better performance, thus applying a kd-tree in searching for k-nearest neighbors of each camera who has three dimensional coordinates should outperform the linear-scan method.

2.7 Image-Based Rendering

Image-based rendering (IBR) is a rendering technique that creates novel views of realistic and synthetic objects using a limited number of images and geometric information. There are three sub-categories in IBR: rendering with explicit geometries, implicit geometries and no geometries.

Model consisting of explicit geometry information such as a set of images of a scene and corresponding depth maps can be rendered from a nearby viewpoint by projecting the pixels of the original image to its 3D coordinates, and re-projecting it onto a new picture of a different point of view. For instance, McMillan et al. (1997) introduced 3D warping, a technique that resolves occluding-artifacts by warping two images and compositing the results.

Rendering techniques relying on no geometric information of objects are based on the plenoptic function (Adelson & Bergen 1991) and its variants. A plenoptic function is a seven-dimensional function describing the intensity distribution of a bundle of rays on the pupil at any location (V_x, V_y, V_z) at every possible angle (θ, ϕ), for every wavelength λ, at every time t, i.e.,

$$P_7 = P(V_x, V_y, V_z, \theta, \phi, \lambda, t). \tag{2.4}$$

As shown in Figure 2.4, by fixing the lighting condition (i.e. dropping the λ parameter), and introducing a static environment (i.e. dropping the time parameter), McMillan and Bishop (1995) introduced the five-dimensional plenoptic function:

$$P_5 = P(V_x, V_y, V_z, \theta, \phi). \tag{2.5}$$

Light field (Levoy & Hanrahan 1996) and lumigraph (Gortler *et al.* 1996) systems re-

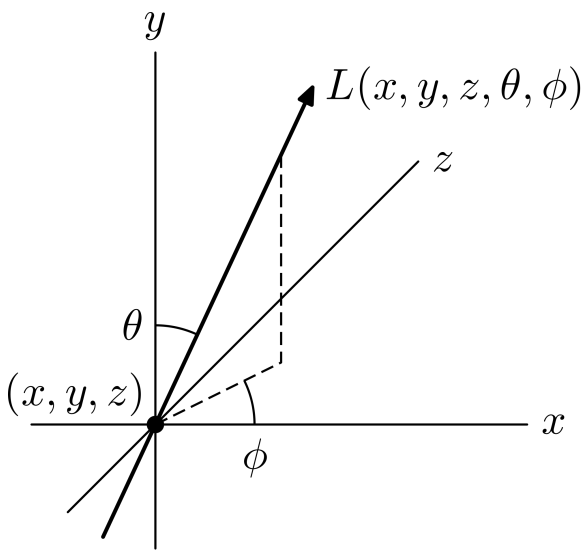


FIG. 2.4. The Five Dimensions of the Plenoptic Function

duce the five-dimensional plenoptic function to four dimensions by making an assumption that the viewpoint stays outside the bounding box of an object:

$$P_4 = P(u, v, s, t). \tag{2.6}$$

So the coordinates on the u-v plane (u, v), and the coordinates on the s-t plane (s, t) can be used to define the light ray (as shown in Figure 2.5).

Shum and He (1999) presented a three-parameter plenoptic function by rotating the camera in concentric circles at varied radius from the rotating axis, so the plenoptic function is determined by the rotation angle, radius and vertical elevation. Chen (1995) and Szeliski and Shum (1997) took one step further: five of the seven parameters of the complete plenoptic function were dropped by fixing the viewpoint, i.e. V_x, V_y and V_z are dropped. So that the adapted plenoptic function becomes two-dimensional, as shown below. In other words, a regular image with limited field of view can be considered incomplete variant of the

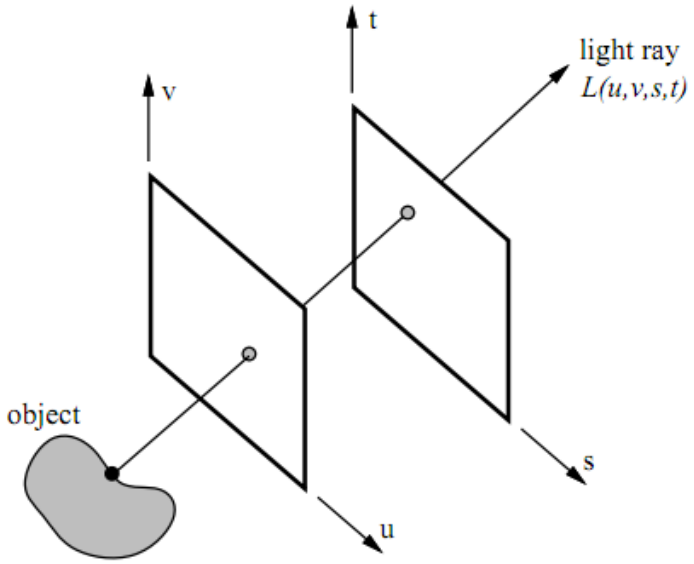


FIG. 2.5. Defining the Light Ray Using u-v and s-t planes

plenoptic function with a fixed viewpoint.

$$P_2 = P(\theta, \phi). \tag{2.7}$$

Chapter 3

THE FRAMEWORK

My framework is designed for point cloud reconstruction from image datasets taken from a fly-by remote-controlled helicopter. The obtained images are then sent to the framework for processing to generate a point cloud representation of the area.

The reconstruction process is decomposed into several stages (Snavely, Seitz, & Szeliski 2006), which include feature extraction, feature matching, scene reconstruction and camera position approximation. Based on the scalability of each stage, parallel solutions are designed using multi-core processors and Graphics Processing Units (GPU). Our framework is presented in Figure 3.1.

In the feature extraction stage (Figure 3.1(a)), image features are extracted using an existing GPU implementation (Wu 2011) of the SIFT algorithm (Lowe 2004) and written back to the host. In the feature matching stage (Figure 3.1(b) and (e)), in order to reduce the image pair comparisons, I introduce an algorithm that reduces the number of images to be matched. For image data obtained in a continuous path with a camera pointed perpendicular to the path, the presented algorithm limits the image correlation to nearby images. The next stage is scene reconstruction and camera position approximation (Figure 3.1(c)). Then a k-nearest neighbor search is performed (Figure 3.1(d)) on the approximated camera positions, and based on the result of the search, for each image i a list of images that are close to i is

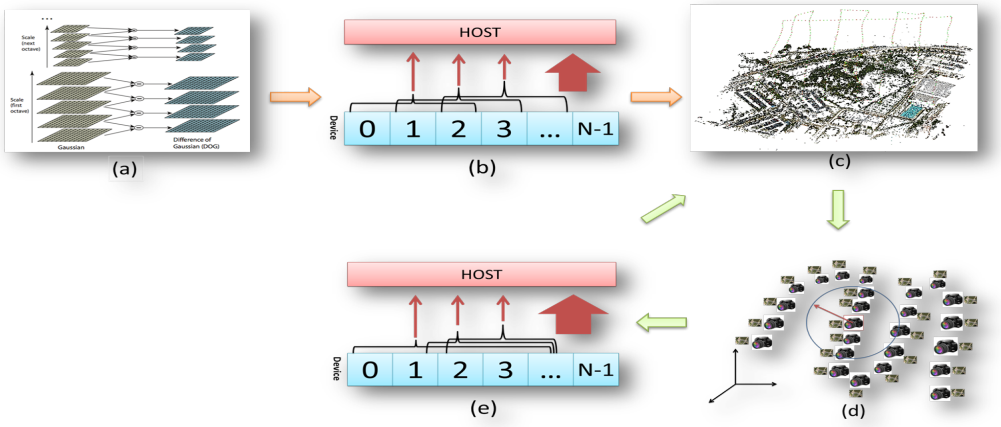


FIG. 3.1. Work flow in the framework: (a) Extract SIFT features; (b) Pairwise feature matching (every consecutive three images); (c) Scene reconstruction and camera position approximation; (d) Find k nearest neighbors of each camera position; (e) 2^{nd} scene reconstruction and camera position approximation; The difference between (b) and (e) is the number of images to be compared: in (b) the number is the same for all the reference images, and in (e) this number varies for different reference images.

passed back to the feature matching stage and matched one more time. From experiments on a NVIDIA GeForce 9800 GT graphics card, the maximum number of consecutive image features copied from the host to the device without causing external fragmentation is 50.

The next stage is performed twice (Figure 3.1(b) and (e)): in (b), the number of query images is a flexible window size, whereas in (e), the number of query images for each reference image is the same.

3.1 Pseudocode

The following is the pseudocode for the feature matching routine. The constant 16 in line 5 was experimentally determined by Snavely, Seitz and Szeliski (2006).

```
1: Decompress SIFT features for each image i
2: for every image i in the image list do
3:    for each image j between image i − 1 and image 0 do
4:       match two sets of key points of image i and image j
5:       if number of matched key points is less than 16 then
6:          break
7:       else
8:          record the matched key points
9:       end if
10:   end for
11: end for
```

3.2 Feature Decompression

After the features are extracted on the GPU, they are written back to the host as compressed files. In order to continue our framework, these compressed features must be decompressed (line 1 in the pseudocode) and be passed to the GPU. The decompression is performed on the CPU in parallel using a generic parallel algorithm, ”parallel-for”, from the Intel Threading Building Blocks (TBB) (Int 2011). The decompression task on the compressed features in the image list is distributed to parallel virtual threads by the TBB task scheduler, as shown in Figure 3.2.

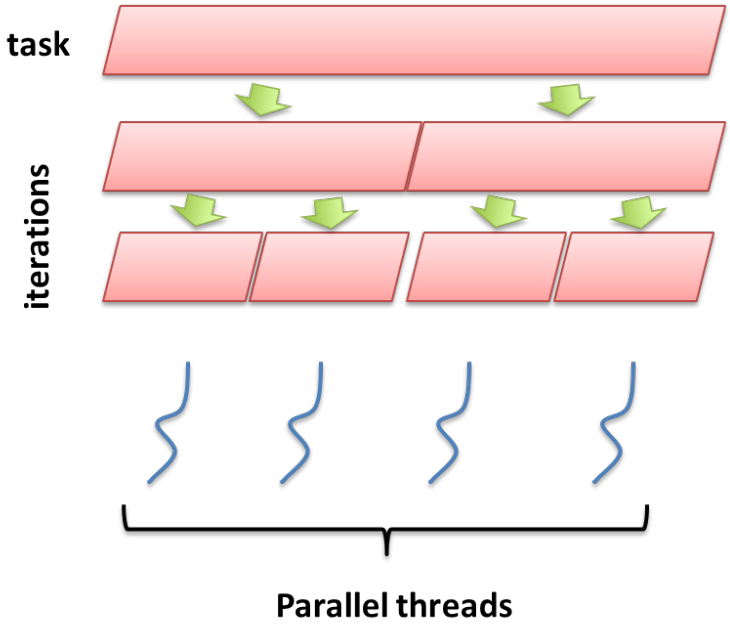


FIG. 3.2. TBB scheduler distributes each iteration of parallel-for to parallel threads

The feature decompression in Bundler (Snavely, Seitz, & Szeliski 2006) is linear, with the compressed features for each image is operated on one by one. In our framework, the TBB task scheduler assigns virtual threads to operate on a segment of the list of files.

In other words, with n parallel virtual threads and a list of N files, each virtual thread decompresses the N/n files in parallel.

3.3 Camera Positions and Image Neighbors

After the features in the images are matched (line 2 through 11 in the pseudocode), scene reconstruction and camera positions are approximated on the CPU (Figure 3.1(c)). The camera positions and the corresponding images from which the cameras were approximated are passed to the GPU and a k-nearest neighbor search algorithm is performed for each camera. As shown in Figure 3.3.

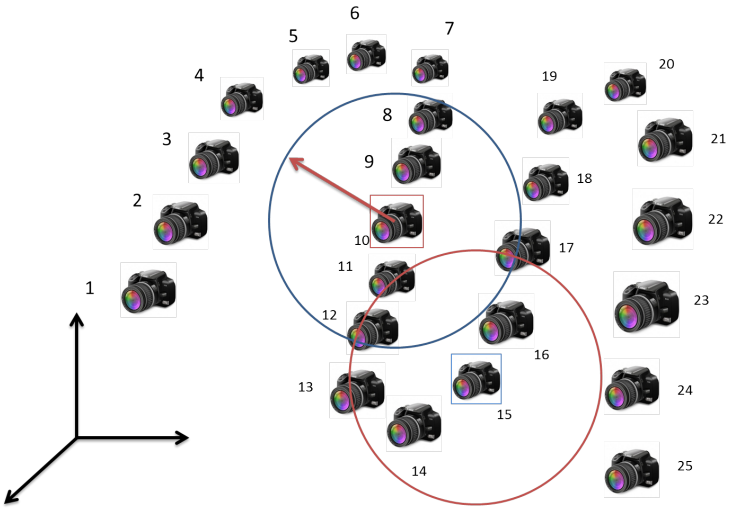


FIG. 3.3. Refinement of Image Matching Base on Camera Positions

For instance, camera 10 and camera 15 in Figure 3.3: in stage (b), image 10 compares to images 8, 9 and image 15 compares to images 12, 13, 14, and after stage (d), based on a number of k neighbors (k = 5, for example) image 10's neighbor list is updated to images 8, 9, 11, 12, 17; image 15's neighbor list is updated to images 12, 13, 14, 16, 17. The

updated neighbor list for each image is sent to the image matching stage and processed again through the framework.

3.4 Analysis of Time Complexity

In this section, I will analyze the time complexity of both image feature matching stages (Figure 3.1(b) and (e)).

3.4.1 Feature Matching: First Time

The original application ((Snavely, Seitz, & Szeliski 2006)) was applied to datasets of unordered images, so features of every two images in the dataset are matched, and the pseudocode of the image feature matching is shown below:

```
1:  for every image i in the image list do
2:    for each image j between image 0 and image i do
3:      match two sets of key points of image i and image j
4:      if number of matched key points is less than 16 then
5:        continue
6:      else
7:        record the matched key points
8:      end if
9:    end for
10: end for
```

Notice the second line of the code, where image i is matched with every other image in the dataset. As a result, the time complexity of the image matching stage can be calculated from the following equation:

$$1 + 2 + 3 + ... + (N - 1) = \frac{N * (N - 1)}{2} = O(N^2) \tag{3.1}$$

On the other hand, our algorithm for this stage of the framework is shown below:

1: **for** every image i in the image list **do**
2: **for** each image j between image $i - 1$ and image 0 **do**
3: match two sets of key points of image i and image j
4: **if** number of matched key points is less than 16 **then**
5: break
6: **else**
7: record the matched key points
8: **end if**
9: **end for**
10: **end for**

In line 4-5 of the pseudocode, the process is stopped by the first image j who has less than 16 features matched to image i. As a result, in the worst case scenario, every image j ($0 \leq$ j $\leq$ i-1) has more than 16 matched features to every other image in the sequence $i - 1$, $i - 2$, $i - 3$, ..., 0, which add up to i images in total. So the theoretical time complexity of this stage is calculated from the following equation:

$$1 + 2 + ... + N - 1 = \frac{N * (N - 1)}{2} = O(N^2) \tag{3.2}$$

We claimed in the previous chapters that our algorithm for feature matching is a linear algorithm, in that except for the taking off and landing area on the navigation path, the helicopter moves along the path at a certain speed, and the field of view (FOV) of the camera can be considered as a constant. For each position where a picture is taken, the length of the path covered by the FOV of this camera is in the range (l_{min}, l_{max}), where

l_{min} and l_{max} are the minimum and maximum length along the navigation path covered by the camera FOV. For a non-periodic path, such as images taken from a moving vehicle, the length of the path (or the number of images along this path that have enough overlap features) is a linear function with regard to the velocity of the moving vehicle; for a periodic path, as the one shown in Figure 1.1, the claim still stands for consecutive images along the path. For images from paths parallel to each other but have overlapping features, the algorithm described in section 3.4.2 is a linear time complexity algorithm that solves this problem.

3.4.2 Feature Matching: Second Time

In the fourth stage (Figure 3.1(d)) in the framework, based on the reconstructed camera positions, the k-nearest neighbor search algorithm is applied to each camera, and its k nearest neighbors are located, thus the images corresponding to the k camera parameters are recorded. As a result, for each image i in the dataset, its k nearest neighbors are considered to be the ones with enough overlap with i to contribute to the reconstruction. Here, the number of neighboring images k should be determined by the density of the datasets. In the experiments during the thesis project, an average of 10 neighbors for each image are used to produce visually satisfactory results.

The feature matching stage is performed a second time, but with a knowledge that for each image i in the dataset, only the k images we located based on the camera positions should be matched. The pseudocode for this stage is the following:

```
1: for every image i in the image list do
2:     for each image j in image i's neighbor list do
3:         match two sets of key points of image i and image j
4:         if number of matched key points is less than 16 then
5:             continue
```

6: **else**
7: record the matched key points
8: **end if**
9: **end for**
10: **end for**

In line 2 of the pseudocode, every image i is compared to its k neighbors, so the time complexity for this stage is the following:

$$k + k + k + ... + k = k * N = O(N) \tag{3.3}$$

As a result, the time complexity of the feature matching stage remains linear.

Chapter 4

PARALLEL ALGORITHMS

In this chapter, the parallel algorithms included in the project are described in detail. I will start with the concept of general-purpose computing on graphics processing units (or GPGPU), NVIDIA's Compute Unified Device Architecture (or CUDA), and then describe the CUDA kernels and their thread distribution in the framework.

4.1 GPGPU and CUDA

4.1.1 General-Purpose Computing on GPUs

A basic graphics pipeline includes three stages: a user program on the CPU supplies primitives (points, lines and polygons) in the form of 3D geometries, and each geometry is identified by a collection of vertices, which are points with coordinates in space; and every vertex in these 3D primitives is processed so that the primitives are transformed into 2D screen triangles; after that, every pixel in the 2D screen triangles is processed into a fragment, which contains necessary data to generate a pixel to be drawn in the framebuffer, where a frame of data would be displayed. OpenGL Shading Language (GLSL) provides operators similar to the ones in the C programming language, and it can be used to implement programs that execute in the graphics pipeline, so non-graphics related data can be assembled as graphics primitives and passed through the graphics pipeline for general

purpose computations.

The traditional functionality of graphics processing units has been limited to accelerate certain parts of the graphics pipeline. They are designed for solving computer graphics problems, so the operations and programming methods are very restrictive. For example, GPUs can only process independent vertices and fragments, but this process can be performed in parallel on many vertices or fragments at the same time. Due to this characteristic, GPUs can be considered as stream processors, which means one kernel, or function executed on a GPU, can be executed on many data that require the same computation. It emphasizes executing thousands of concurrent threads relatively slowly, rather than executing a single thread quickly (like CPUs).

GPGPU is a technique of using a GPU to perform computation traditionally handled by the CPUs. Additional programmable stages and higher precision arithmetic added to the rendering pipeline allows programmers to use stream processing on non-graphics data for general purpose computations.

4.1.2 Compute Unified Device Architecture (CUDA)

CUDA was introduced in 2007 by NVIDIA to give programmers access to the virtual instruction set and memory of the parallel computational elements in CUDA-enabled GPUs.

The CPU and its main memory are referred to as the host, and GPU and its memory are referred to as the device. Basic CUDA processing flow includes four major steps: a) Copy data from the main memory to GPU memory. b) a CUDA kernel is called on the host which instructs the process on GPU. c) the GPU executes the kernel in parallel in each core. d) Copy the result from device memory to the host memory.

The CUDA runtime launches kernels on a compute grid architecture, each of which consists of parallel blocks, and each block comprises parallel threads.

Table 4.1. GPU Memory Type, Size, Speed, Sharing Scope and Lifetime

Memory	Size	Speed	Sharing Scope	Lifetime
register	small	Very fast	thread	kernel
local	small	Fast	thread	kernel
shared	small	Fast	block	kernel
global	large	Slow	grid	Application
constant	small	Fast	grid (read only)	Application

There are five categories of memory on the GPU: register, local memory, shared memory, global memory and constant memory. Table 4.1 shows the size, access speed, share condition and lifetime for each type of memory. Registers and local memory are allocated to each individual thread, they are fast but not visible to any other thread; shared memory are allocated to all the threads in the same block, it is fast to access, but has limited size; global memory has large capacity but great latency to access; constant memory is for storing read-only data, with high parallelism but low capacity.

Besides all these types of device memory, programmers are allowed to allocate part of the global memory as texture memory if data access pattern has considerable spatial locality. When consecutive threads access consecutive memory addresses, it is called coalesced read, and its advantage is that we get the same speed up as the number of consecutive threads when reading the consecutive memory over sequential read operations.

4.2 Work Flow in the Framework

There are five stages in the framework: SIFT feature extraction, image feature matching, scene reconstruction, finding neighboring images based on the reconstructed camera positions and repeat image feature matching and scene reconstruction. In the current framework, SIFT feature extraction, image feature matching and finding neighboring images stages are operated on the GPU whereas the scene reconstruction stage is performed on the

CPU (as shown in Figure 4.1).

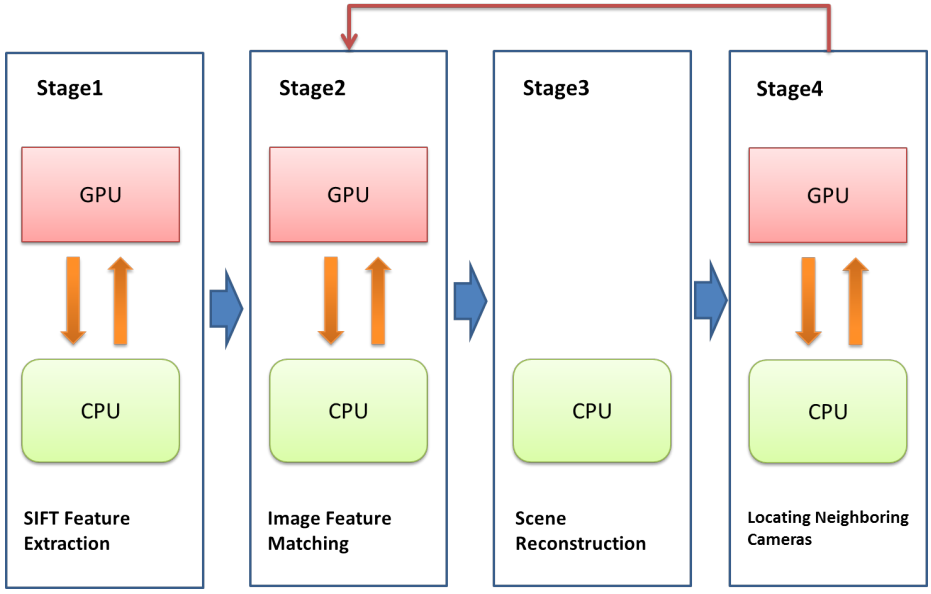


FIG. 4.1. Stages in the Framework

4.3 Kernels and Threads Distributions

4.3.1 K-nearest neighbor search algorithm

The core implementation of k-nearest neighbor search algorithm is adopted from Garcia et al. presented a CUDA implementation of the two-step brute force method (Garcia, Debreuve, & Barlaud 2008): 1) Compute the distance between each reference point and the query point; 2) Sort the distances and return the first k candidates.

Two kinds of GPU memory are used: global memory and texture memory. The global memory has large bandwidth, but in order to achieve high performance, operations with coalesced access to the global memory should be performed in global memory, so the sorting algorithm is implemented by accessing the global memory (coalesced read operations). On

the other hand, the computation of reference-query point distance is performed using the texture memory because there are fewer penalties in the texture memory for non-coalesced read operations.

After the search algorithm, the k nearest neighbors of each query point and the distance between the query point and its k nearest neighbors are returned.

4.3.2 Match Two Images

First we need to introduce some notations: f_{i1} is a feature (128 dimensional descriptor) of image i, f_{j1} and f_{j2} are features of image j and the nearest neighbor and next nearest neighbor of f_{i1}, respectively; d_1 is the Euclidean distance between f_{i1} and f_{j1}, and d_2 is the Euclidean distance between f_{i1} and f_{j2}.

The image feature matching stage takes in features of image i (reference points) and image j (query points), locates the k-nearest neighbors of each query point among the reference points and returns the results in an array as shown in Figure 4.2: since we need only the nearest and the next nearest neighbor of each query point when matching two images, the array consists of $d_{1[0:n-1]}$ and $d_{2[0:n-1]}$, where d_{1*} is distance between query point and its nearest neighbor in image i, d_{2*} is the distance between query point and its next nearest neighbor in image i, and n is the number of query points.

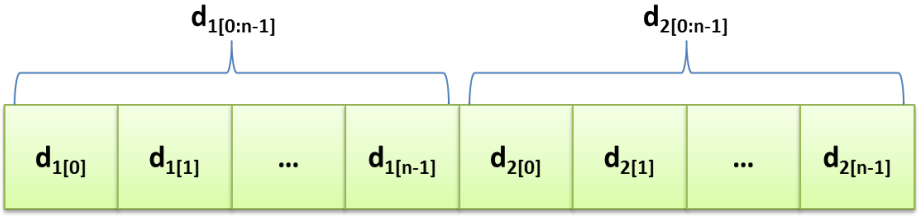


FIG. 4.2. Array of Distances to Nearest Neighbor and Next Nearest Neighbor

According to Lowe (2004), two features are considered as a match iff the equation 4.1 is true, where $ratio$ is a number determined by experiments, and Snavely et al. (2006) recommended 0.36.

$$\frac{d_{1*}}{d_{2*}} \leq ratio \tag{4.1}$$

A CUDA kernel is designed to perform this validation check on the ratio between d_{1*} and d_{2*} of every query point: if the equation 4.1 is true, a flag is turned on in an array of flags that is located in the shared memory of each block. This kernel is launched by 512 threads/block, and 16 blocks/grid. The following is the pseudocode for this kernel:

1: declare a data structure $cache$ in shared memory to store match information
2: assign thread id tid to a thread in current block
3: **while** $tid <$ # query points **do**
4: **if** $\frac{d_{1[tid]}}{d_{2[tid]}} \leq ratio$ **then**
5: increment counter for matched query point
6: **end if**
7: increment tid of every thread in the grid to process data in the next grid
8: **end while**
9: thread tid updates corresponding entry in $cache$
10: synchronize threads in the same block
11: use a while-loop for a summation-reduction on $cache$

Line 8 in the pseudocode, every entry of $cache$ in the shared memory is set up with the number of matched features each thread found, then line 10, a summation-reduction is performed on this data structure of entries. In Figure 4.3, one step of a summation reduction is presented. After this reduction, the first entry in $cache$ would be the number of matches found by all the threads in one block, and $cache$ is returned to the host for summation to

get total number of matches between the two images.

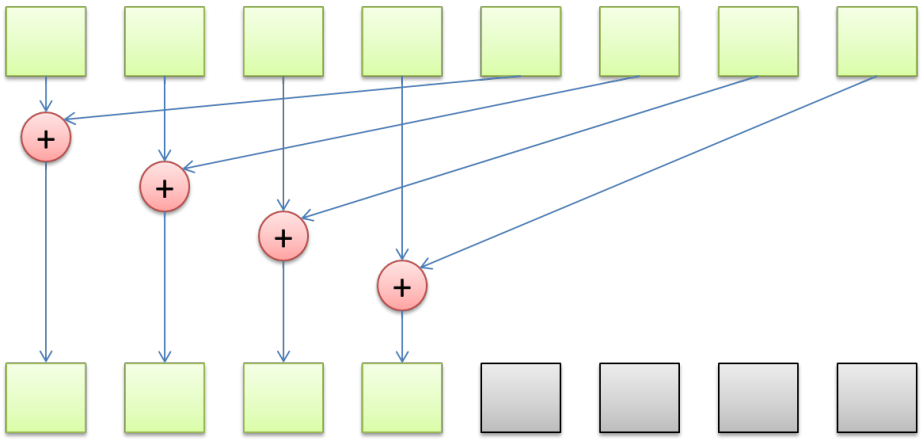


FIG. 4.3. Summation reduction on the *cache* in the shared memory

Chapter 5

RESULTS

This chapter presents the performance of the new framework.

5.1 Feature Extraction

Image features are extracted using a GPU implementation of the SIFT algorithm (Lowe 2004) by Wu (2011), and the Figure 5.1 is the experiment result provided with the implementation.

From Figure 5.1, the CUDA implementation provides better performance than other implementations when the resolution of the image is low, i.e. 320*240; however, in our framework, the feature extraction is performed by the GLSL implementation of the algorithm because it can be applied to a wide range of image resolutions.

Table 5.1 shows the execution time of the feature extraction stage on several image datasets with 1824*1368 resolution. The images we used are consecutive images from the same dataset of 700 images: first 100, 200, 300 images, etc. Figure 5.2 is a plot of the execution time for Bundler and the new framework: the execution time for feature extraction stage in Bundler is between 2000 seconds and 11000 seconds, whereas the new framework takes between 200 seconds and 2000 seconds.

Figure 5.3 shows the speedup of feature extraction on the datasets. The plotted line

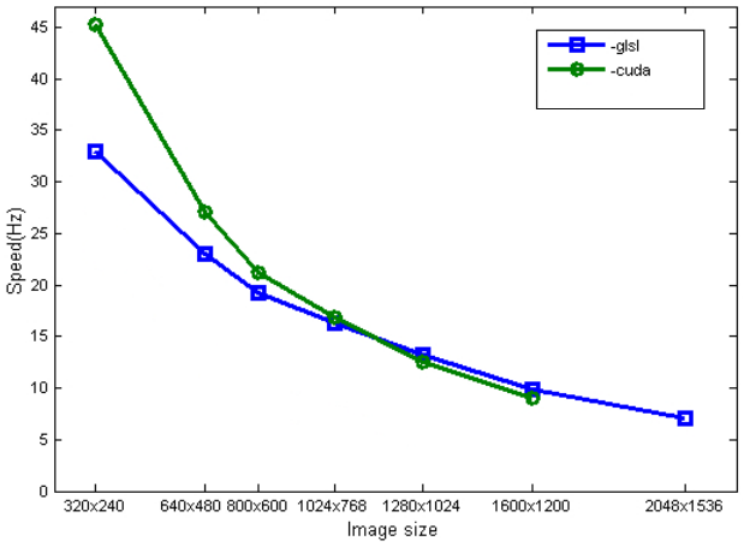


FIG. 5.1. Speed chart for the GPU SIFT Implementation

Table 5.1. Execution Time on Feature Extraction (unit: HH:MM:SS)

# images in the dataset	Bundler	Our Framework
100 images	00:30:37	00:04:31
200 images	01:01:55	00:09:28
300 images	01:28:15	00:13:48
400 images	01:55:30	00:18:07
700 images	03:31:39	00:33:00

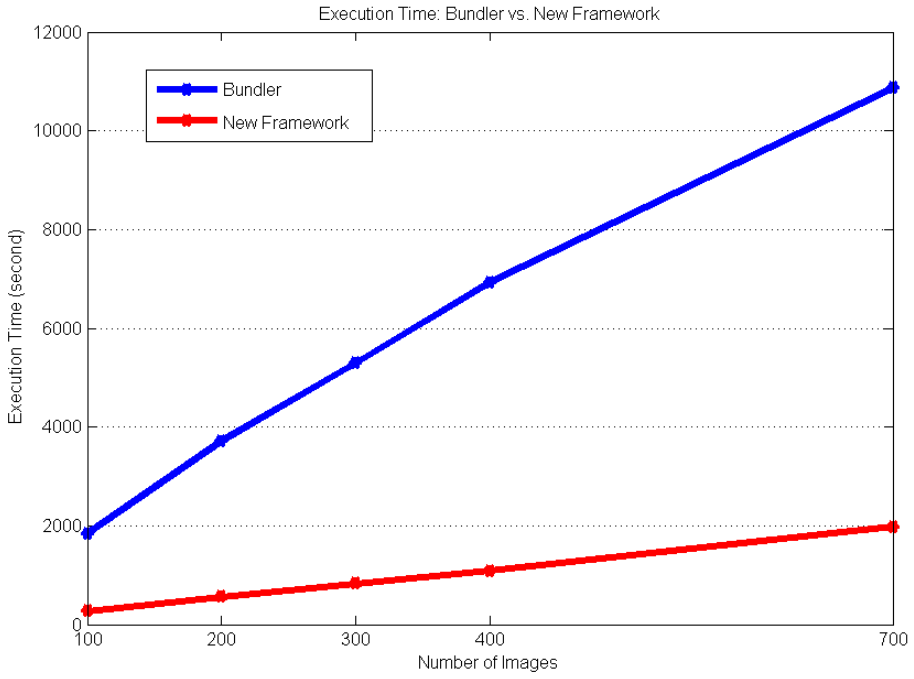


FIG. 5.2. Execution Time of Feature Extraction Stage

is almost flat, because the execution time is determined merely by the number of images: image features are extracted from one image at a time. In addition, the speedup of the GPU solution is stable across multiple image datasets with varied number of images, although the features are transfered between CPU and GPU, the communication cost is not affecting the overall speedup.

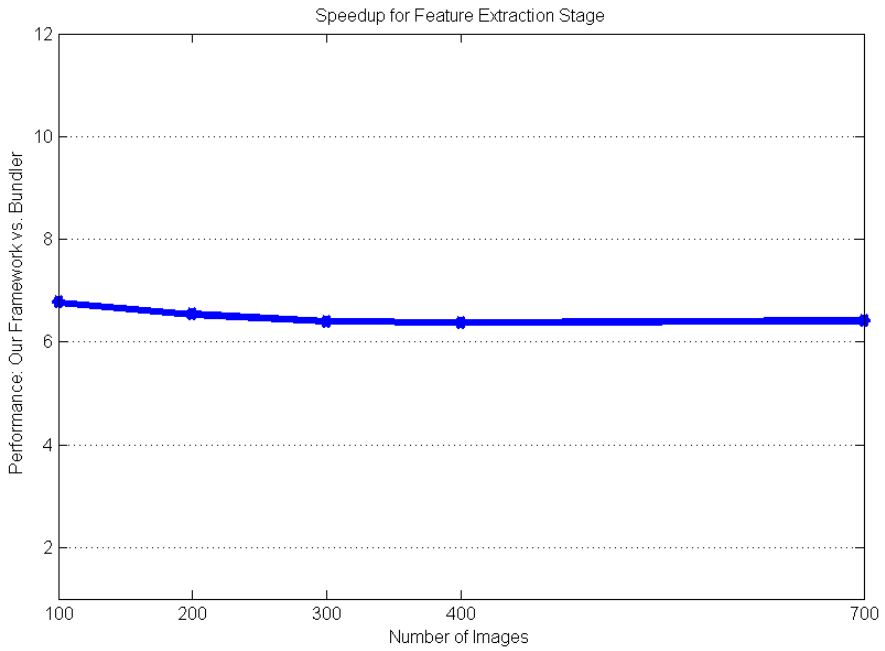


FIG. 5.3. Speedup of Feature Extraction Stage

5.2 Feature Matching

We used images of 1824*1368 resolution during the experiments, and the execution time for feature matching stage is shown in Table 5.2, and Figure 5.4(a), Figure 5.4(b) are plots of the execution time of Bundler and the new framework respectively. The speedup

Table 5.2. Execution Time on Feature Matching (unit: HH:MM:SS)

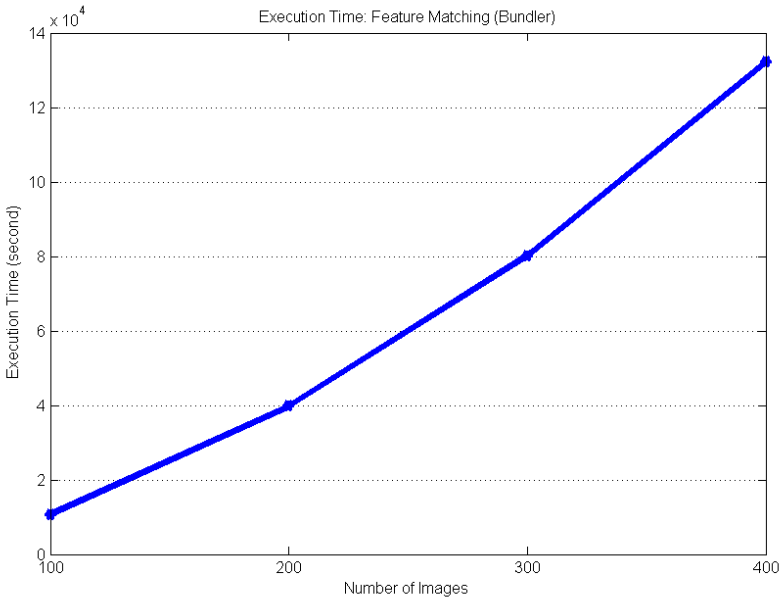
# images in the dataset	Bundler	Our Framework
100 images	2:59:22	00:02:40
200 images	11:02:38	00:06:34
300 images	22:16:23	00:09:41
400 images	36:42:26	00:14:08

of our design against Bundler's feature matching stage is shown in Figure 5.4.

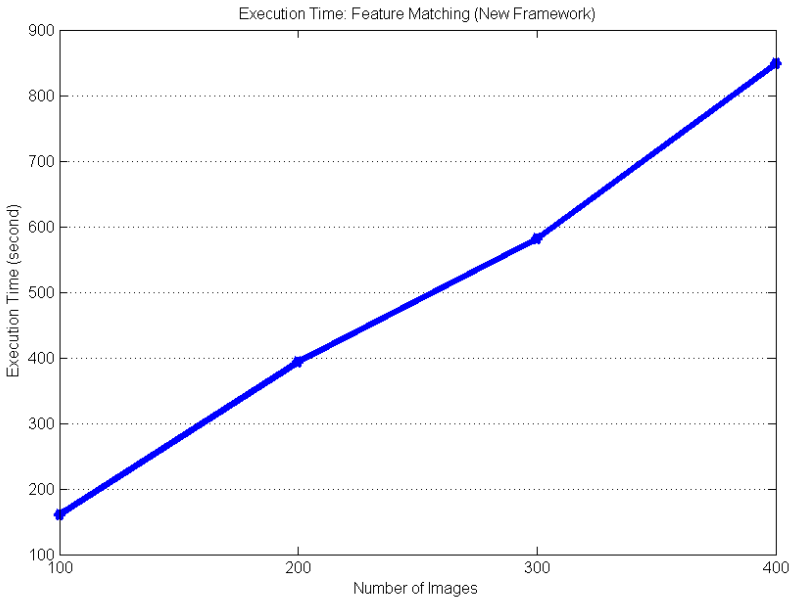
From Figure 5.4, the speedup of the feature matching stage of our framework over Bundler increases as the number of images in the dataset increases, in that Bundler matches features of every two images in the datasets, whereas flexible window size algorithm takes advantage of the fact that only limited images in a sequence are needed for matching to one image. So the execution time of the new framework in feature matching stage is a linear function about the number of images in the dataset, while the execution time of Bundler in feature matching stage is a function about the square of the number of images, which is why the speedup increases as the number of images increases.

5.3 Scene Reconstruction and Camera Position Approximation

We kept the bundle adjustment algorithm on the scene reconstruction and camera approximation from Bundler (Snavely, Seitz, & Szeliski 2006), but since we are using a different nearest neighbor search algorithm in the feature matching stage, i.e. linear scan or brute force method. The matched features are different from those we get from the feature matching stage of Bundler. According to the reconstructed point cloud from both applications, our framework delivers better visual correctness. Figure 5.5(a) is the side view of a reconstructed point cloud from 400 image dataset, and Figure 5.5(b) shows the side view of the point cloud by Bundler from the same dataset. From Figure 5.5(a) and Figure 5.5(b), we find that the structure in our point cloud is visually correct, since no feature points are



(a) Execution Time of Feature Matching Stage: Bundler (unit: 10^4 seconds)



(b) Execution Time of Feature Matching Stage: New Framework

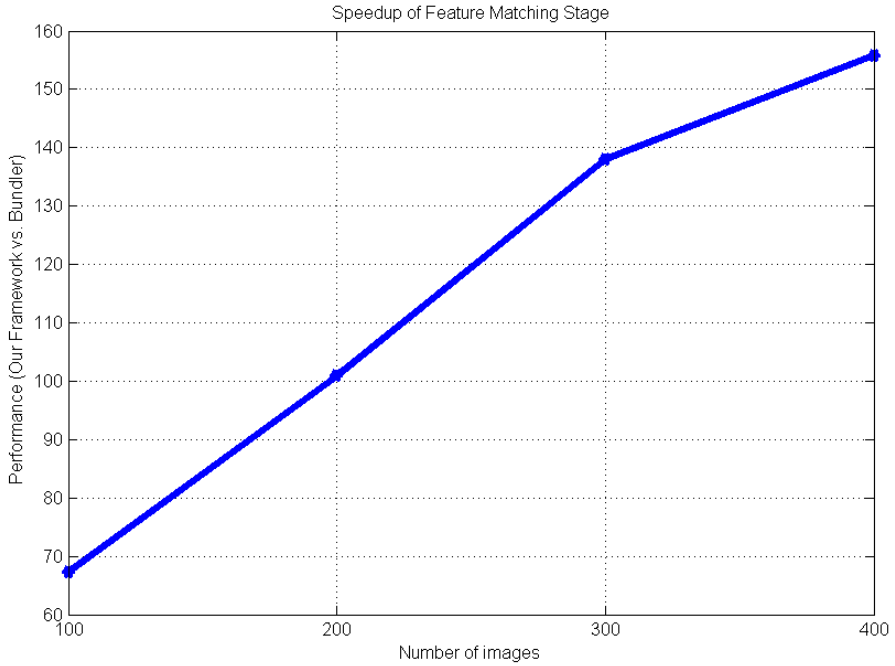
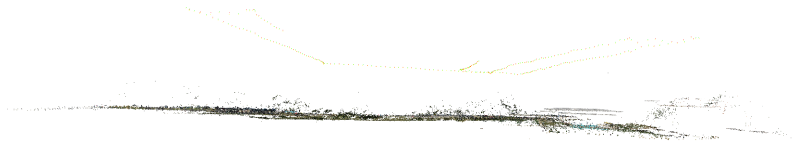
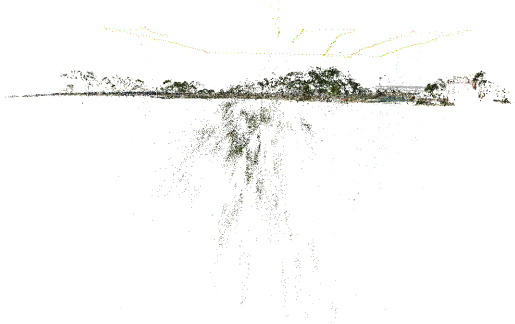


FIG. 5.4. Speedup of the feature matching stage

placed below the horizon, whereas the point cloud generated by Bundler has a large number of points placed below the horizon. This incorrect reconstruction is the main reason that Bundler's execution takes extremely long since it takes more iterations for the non-linear least square optimization to converge.



(a) Side view of point cloud generated by our framework



(b) Side view of point cloud generated by Bundler

Table 5.3 shows the execution time on scene reconstruction stage. Figure 5.5 is a plot showing the execution time of Bundler and the new framework.

In Figure 5.6, the speedup of this stage decreases as the number of images in the datasets increases. The reason for this result is that the scene reconstruction process is not

Table 5.3. Execution Time on Scene Reconstruction (unit: HH:MM:SS)

# images in the dataset	Bundler	Our Framework
100 images	01:37:46	00:02:14
200 images	07:05:03	00:11:28
300 images	09:22:03	01:06:59
400 images	27:04:50	04:03:28

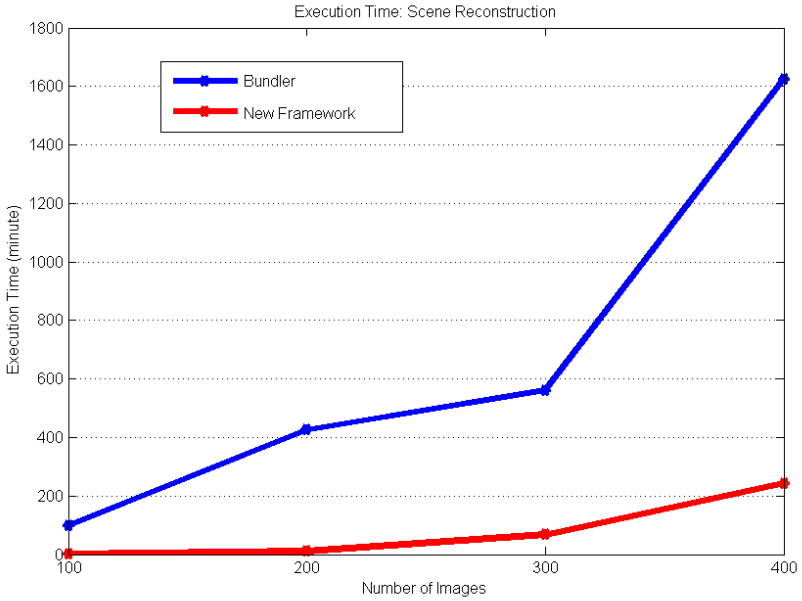


FIG. 5.5. Execution time for Scene Reconstruction

optimized, and as both our framework and Bundler are reconstructing the scene using CPU implementation of the optimization algorithms.

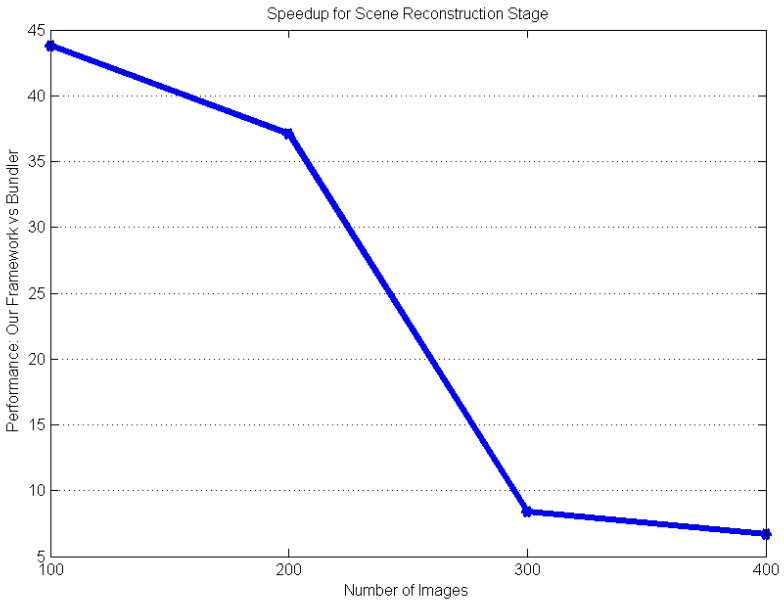


FIG. 5.6. Speedup of the scene reconstruction stage

5.4 Overall Speedup

Table 5.4 presents the overall speedup of the parallel framework over the Bundler application.

5.5 More Reconstructions

In this section, the accumulation of reconstructed points of the same forest site is presented in the Figure 5.5 and Figure 5.5.

Table 5.4. Speedup over Bundler

# images in the dataset	Speedup
100 images	32.68
200 images	41.80
300 images	21.96
400 images	14.30

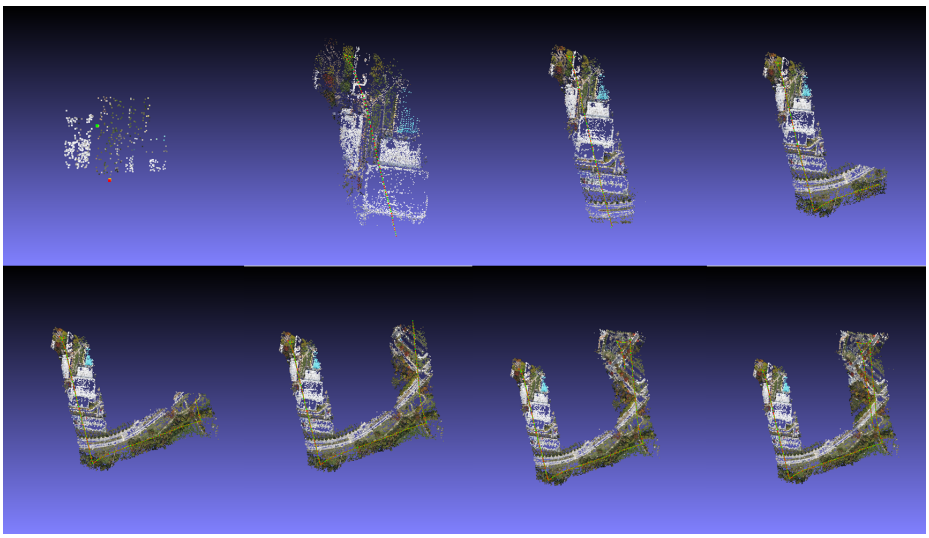


FIG. 5.7. Reconstruction accumulation: first half

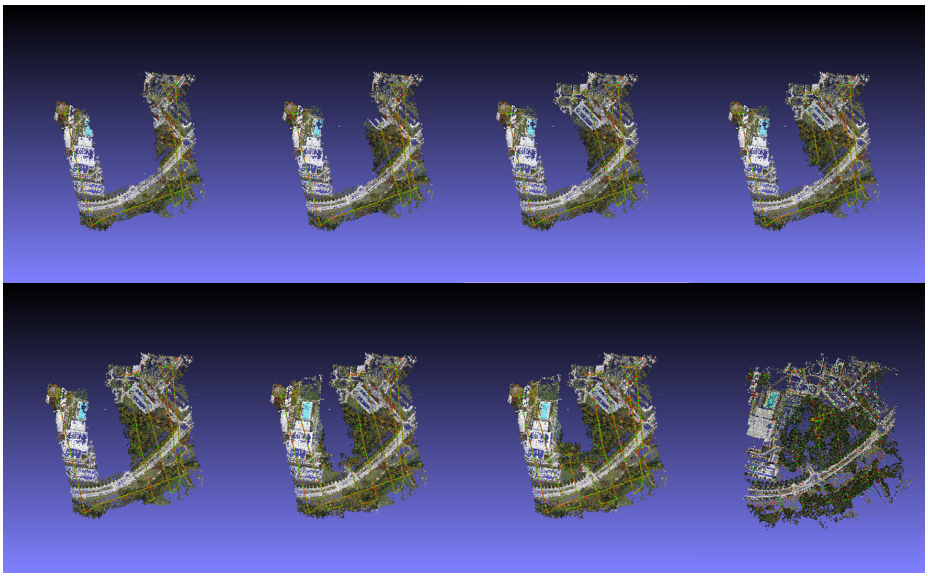


FIG. 5.8. Reconstruction accumulation: second half

Chapter 6

QUALITY EVALUATION

In this section, a method is proposed to evaluate the quality of the reconstructed point clouds, i.e. difference between the camera positions recovered by both Bundler and by the new framework.

6.1 Reconstructed Camera Positions

Given two points in space, V_1 (v_{1x}, v_{1y}, v_{1z}) and V_2 (v_{2x}, v_{2y}, v_{2z}), the Euclidean distance between V_1 and V_2 is:

$$\sqrt{(v_{1x} - v_{2x})^2 + (v_{1y} - v_{2y})^2 + (v_{1z} - v_{2z})^2} \tag{6.1}$$

The image datasets are obtained with the cameras pointing at the area of interest, perpendicular to the path. Therefore, the quality of recovered cameras positions can be considered as one aspect of the evaluation metric. We will be considering cameras recovered by both Bundler and the new framework and evaluate the differences between the positions of the cameras in space using the *Root-Mean-Square* (RMS) method.

For two cameras recovered using the same photograph by both Bundler and the new framework, C_k (x_k, y_k, z_k) and C'_k (x'_k, y'_k, z'_k), the vector $\overrightarrow{C_k C'_k}$ is the difference between the two recovered cameras. Projecting $\overrightarrow{C_k C'_k}$ along $\overrightarrow{C_{k-1} C_{k+1}}$ and perpendicular to

$\overrightarrow{C_{k-1}C_{k+1}}$, and measuring the length of both vectors, we get the difference of camera positions recovered from the same photographs by two methods along the navigation path and perpendicular to the path.

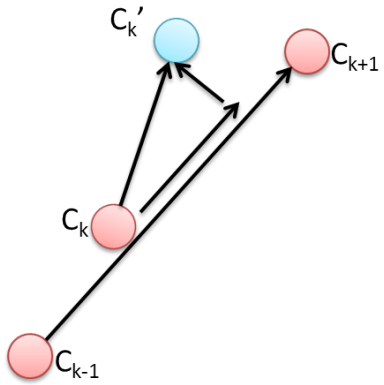


FIG. 6.1. Illustration of difference measurment

Figure 6.2, 6.3 and 6.4 show the differences between recovered camera positions along the path and perpendicular to the path when processing four datasets each with one hundred photos.

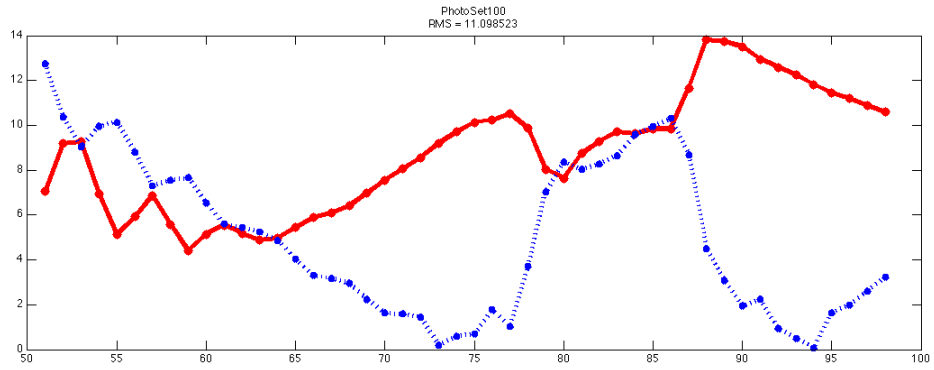


FIG. 6.2. Difference along and perpendicular to the path on a one-hundred-image dataset 1

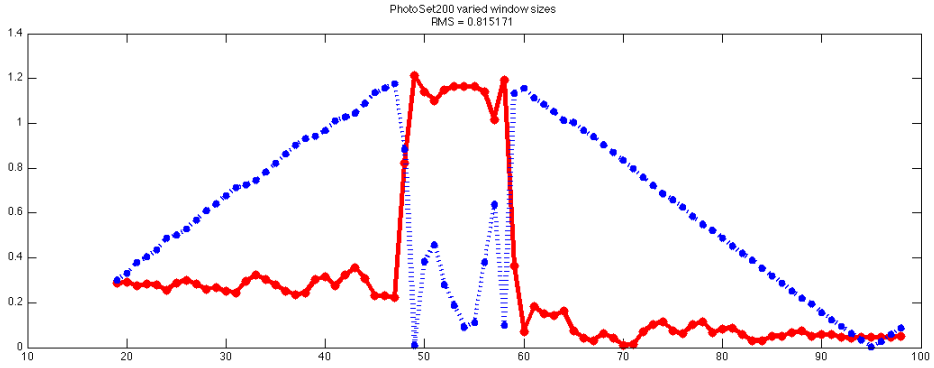


FIG. 6.3. Difference along and perpendicular to the path on a one-hundred-image dataset 2

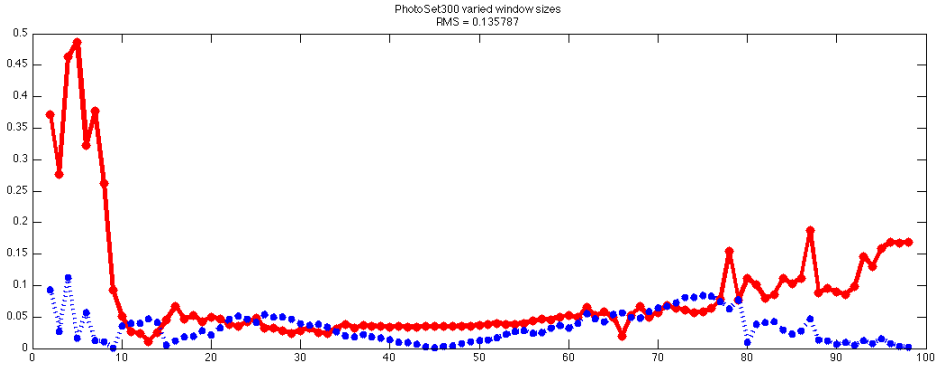


FIG. 6.4. Difference along and perpendicular to the path on a one-hundred-image dataset 3

Chapter 7

CONCLUSION AND FUTURE WORK

In this thesis project, I designed parallel solutions for the structure-from-motion application, Bundler (Snavely, Seitz, & Szeliski 2006). Based on the different parallelism of each stage, both CPU and GPU solutions are applied to the framework. According to the experiment results, the performance of all the stages are significantly improved.

The framework in this thesis project performs SIFT feature extraction and image feature matching on the GPU, but not scene reconstruction. The scene reconstruction stage uses bundle adjustment, a non-linear least squares optimization algorithm. One item for future work is to implement this stages on the GPU as well.

In the current framework, data are repeatedly copied between the GPU and CPU memory, which causes certain data transfer overhead. For example, image features are copied back to the CPU after SIFT features are extracted, and later the same features are copied back to the GPU in the feature matching stage. Although our framework delivers good performance in the experiments, this repeating copy operation might cause external fragmentation on the GPU memory and incur a data transfer performance cost. As a result, an important future work would be porting all the stages onto the GPU without repeating data transfer between the CPU and GPU. In other words, for an image dataset, image features should be stored in the global memory of the GPU after feature extraction stage for all the

parallel threads from all the blocks to access in later stages.

In addition, the resolution of the image dataset I used in the project is 1824*1368, and more experiments are needed to determine the optimal image resolution in the dataset. We expect this optimal size to be smaller than the current images. Smaller image sizes would allow colleagues at the geography department collect more images at lower resolution on a storage card in the digital camera during one RC-helicopter fly mission. Also, smaller images would decrease the data processing time, and allow more images to fit into GPU memory, reducing the need to page data from the CPU to GPU.

REFERENCES

[1] Adelson, E. H., and Bergen, J. R. 1991. The plenoptic function and the elements of early vision. In Landy, M. S., and Movshon, A. J., eds., *Computational Models of Visual Processing*. Cambridge, MA: MIT Press. 3–20.

[2] Agarwal, S.; Furukawa, Y.; Snavely, N.; Curless, B.; Seitz, S. M.; and Szeliski, R. 2009. Building rome in a day.

[3] Arya, S.; Mount, D. M.; Netanyahu, N. S.; Silverman, R.; and Wu, A. Y. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM* 45(33):891–923.

[4] Canny, J. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 8:679–698.

[5] Chen, S. E. 1995. Quicktime vr: an image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, 29–38. New York, NY, USA: ACM.

[6] Cornelis, N.; Cornelis, K.; and Gool, L. V. 2006. Fast compact city modeling for navigation pre-visualization. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 1339–1344.

[7] Dandois, J. P., and Ellis, E. C. 2010. Remote sensing of vegetation structure using computer vision. *Remote Sensing*.

[8] Farenzena, M.; Fusiello, A.; and Gherardi, R. 2009. Structure-and-motion pipeline on a hierarchical cluster tree.

[9] Fitzgibbon, A. W., and Zisserman, A. 1998. Automatic camera recovery for closed or open image sequences. In *Proc. European Conf. on Computer Vision*, 311–326.

[10] Garcia, V.; Debreuve, E.; and Barlaud, M. 2008. Fast k nearest neighbor search using gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, 1–6.

[11] Gortler, S. J.; Grzeszczuk, R.; Szeliski, R.; and Cohen, M. F. 1996. The lumigraph. In *Proc. SIGGRAPH*, 43–54.

[12] 2011. Intel threading building blocks http://threadingbuildingblocks.org/.

[13] Kampe, T. U.; Johnson, B. R.; Kuester, M.; and Keller, M. 2010. Neon: the first continental-scale ecological observatory with airborne remote sensing of vegetation canopy biochemistry and structure. In *Journal of Applied Remote Sensing*, volume 2.

[14] Kruppa, E. 1913. Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung.

[15] Levoy, M., and Hanrahan, P. 1996. Light field rendering. In *Proc. SIGGRAPH*, 31–42.

[16] Longuet-Hinggins, H. 1981. A computer algorithm for reconstructing a scene from two projections. *Nature* 293:133–135.

[17] Lowe, D. G. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.

[18] McMillan, L., and Bishop, G. 1995. Plenoptic modeling: An image-based rendering system. In *Proc. SIGGRAPH*, 39–46.

[19] McMillan, L.; Jr.; and Pizer, R. S. 1997. An image-based approach to three-dimensional computer graphics. Technical report.

[20] Microsoft. 2010.

[21] Nocedal, J., and Wright, S. 2000. *Numerical Optimization*. Springer.

[22] Pollefeys, M.; van Gool, L.; Vergauwen, M.; Verbiest, F.; Cornelis, K.; Tops, J.; and Koch, R. 2004. Visual modeling with a hand-held camera. In *Int. J. of Computer Vision*, volume 59(3), 207–232.

[23] Shum, H.-Y., and He, L.-W. 1999. Rendering with concentric mosaics. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, 299–306. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

[24] Snavely, N.; Seitz, S. M.; and Szeliski, R. 2006. Photo tourism: Exploring image collections in 3d. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*.

[25] Szeliski, R., and Shum, H.-Y. 1997. Creating full view panoramic image mosaics and environment maps.

[26] Triggs, B.; McLauchlan, P. F.; Hartley, R. I.; and Fitzgibbon, A. W. 2000. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, number 75 in ICCV '99, 298–372. London, UK: Springer-Verlag.

[27] Weber, R.; Schek, H.-J.; and Blott, S. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, 194–205. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[28] Wu, C. 2011. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). http://cs.unc.edu/ ccwu/siftgpu, April 6, 2011.

[29] Zach, C. 2011. Structure-and-motion (sfm) software.