Geoinformatica (2016) 20:859–878
DOI 10.1007/s10707-016-0253-2

The index array approach and the dual tiled similarity algorithm for UAS hyper-spatial image processing

Lihong Su[1] · **Yuxia Huang**[2] · **James Gibeaut**[1] · **Longzhuang Li**[2]

Received: 8 December 2014 / Revised: 12 November 2015 /
Accepted: 23 March 2016 / Published online: 2 April 2016

Abstract Unmanned aerial systems (UAS) have been used as a robust tool for agricultural and environmental applications in recent years. Remote sensing systems based on UAS typically acquire massive hyper-spatial images in its short turnaround. This paper takes advantage of graphics processing unit (GPU) massive parallel computation in order to process the huge data timely and efficiently. More specifically, this paper presents an index array approach for lens distortion correction and geo-referencing. They are the two essential components in UAS hyper-spatial image processing. The index array approach is also capable of parallelizing image file I/O and the orthoimage generation. In addition, this paper presents the dual tiled similarity algorithm for the image co-registration. The index array approach and the dual tiled similarity algorithm were evaluated using two UAS remote sensing datasets of South Padre island shorelines. The results show that this index array approach was able to speed up at least 10 times the lens distortion correction and the geo-referencing relative to the central processing unit (CPU) computation. This dual tiled algorithm could provide 12 times speedup compared with the CPU similarity computation.

Keywords GPU algorithm · Remote sensing · Unmanned aerial systems · Lens distortion correction · Geo-referencing · Co-registration

✉ Lihong Su
su.lihong@tamucc.edu

[1] Harte Research Institute for Gulf of Mexico Studies, Texas A&M University-Corpus Christi, Corpus Christi, TX, USA

[2] School of Engineering and Computing Sciences, Texas A&M University-Corpus Christi, Corpus Christi, TX, USA

1 Introduction

Unmanned aerial systems (UAS) are experiencing the greatest near-term growth in civil and commercial operations due to their versatility and relatively low initial cost and operating expenses [1]. In recent years UAS have also been used as a robust tool for remote sensing of agricultural and environmental applications. The images acquired with UAS remote sensing usually have a small footprint, hyper-spatial resolution (namely sub-decimeter), and high overlapping ratio due to its low flight altitude and high definition cameras. The small footprint and high overlapping ratio could run in thousands hyper-spatial images for a regular study area, such as the crop field of a farm or the coastal shoreline of an island. The hyper-spatial resolution also means that each individual image has large volume. The large number of large images cause a significant computing load. For example, Texas A&M University-Corpus Christi's RS-16 UAS could produce 200GB imagery by 1 h data acquisition.

The goal of this paper is to efficiently generate an extensive mosaic image that completely covers a regular study area from UAS remote sensing. Four main processes are included in this task. They are 1) lens distortion correction, 2) geo-referencing, 3) co-registration, and 4) mosaicking. These processes typically are computing intensive due to the large amount of data and the complexity of the processes. In addition, for near-real time agricultural and environmental applications, for example, crop health and precision management of pesticides, and flooding detection and relief management, the researchers usually need to see the classification maps derived from the huge volume imagery in a short time, such as the next day. Traditional sequential computing approach is difficult to meet the needs of this time sensitive demand of huge computing intensive tasks.

Big data volume and time-demanding time limit are the two main challenges that the UAS remote sensing faces. The graphics processing unit (GPU) massive parallel computation provides an opportunity of addressing these issues [2, 3]. Some research has been done in the GPU-based processing of remote sensing imagery. For example, references [4–12] explored hyperspectral image processing by the GPU with a focus on handling hundred spectral bands on the same image pixel. References [13–18] proposed the GPU methods for geo-correction and orthorectification (also called geo-referencing) for imagery acquired by UAS commercial off-the-shelf cameras, an airborne pushbroom imager, and high resolution satellite sensors. Some investigators used the GPU to accelerate computation of radiative transfer model [19, 20], and segmentation and classification of remotely sensed imagery [21–23].

This paper explores GPU methods for UAS hyper-spatial remote sensing image processing. More specifically, an index array approach is for lens distortion correction and geo-referencing. This approach is also capable of parallelizing image file I/O and the orthoimage production. In addition, a dual tiled similarity algorithm is for the image co-registration in order to mosaic the images faster.

2 Background

2.1 Principle and workflow of UAS image processing

To generate a mosaic image from original UAS images, typically the following four components are included: lens distortion correction, geo-referencing, co-registration and mosaicking images (Fig. 1). The first two processes are to generate the destination images with geographic

coordinates, while the last two processes are to generate the mosaic image. Each process is briefly explained as follows.

2.1.1 Lens distortion correction

The camera lens distortion correction uses the internal geometry of a camera existing at the time of data capture to transform the photo pixel coordinate system to the imaging plane coordinate system [24]. The internal geometry includes focal length and lens distortion coefficients. There are two types of radial lens distortion that exist: radial distortion Δr and tangential distortion Δt. Since the tangential lens distortion is usually much smaller in magnitude than the radial lens distortion, it is considered negligible. For a digital camera, radial distortion usually is an inward displacement of a given image point from its ideal location. The inward displacement should be eliminated by adding matching displacement on each pixel. The effects of radial lens distortion throughout an image can be approximated using a polynomial [24]. A forward correction model of non-metric digital camera distortion is given as follows [25]:

$$\Delta x = (x - x_0)\left(k_0 r + k_1 r^3 + k_2 r^5\right) \tag{1}$$

$$\Delta y = (y - y_0)\left(k_0 r + k_1 r^3 + k_2 r^5\right) \tag{2}$$

where x_0, y_0 are principal point offsets from image center; x, y are the coordinates of the image point; Δx, Δy are the correction of the image point coordinates. k_0, k_1, k_2 are the radial distortion; and $r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$.

2.1.2 Geo-referencing

Geo-referencing establishes the location of an image in terms of ground coordinate systems. In other words, geo-referencing generates a destination image that has geographic coordinates from a photo (namely an original image) that has no geographic coordinates. This is a key process to make UAS imagery useful for mapping and analysis. The ground coordinate (X, Y, Z) is typically defined as a three-dimensional Cartesian coordinate system, in which (X, Y) utilizes a known map projection such as the Universal Transverse Mercator (UTM) and the Z value is the elevation above the mean sea level by a given vertical datum.

Geo-referencing can be empirically and manually completed through affine transformation between distorted photo coordinates (column, row) and map coordinates (X, Y) with well-

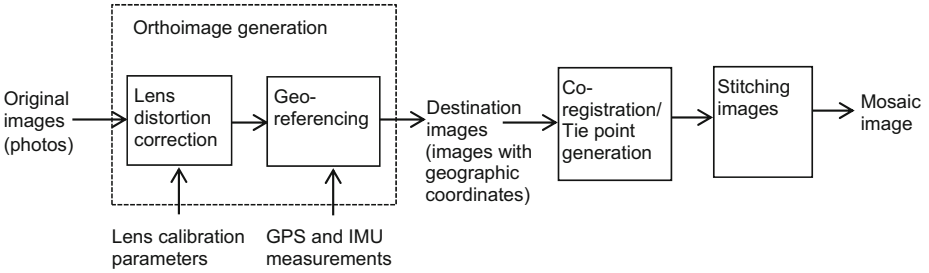

Fig. 1 Workflow of UAS remote sensing image processing

distributed ground control points (GCPs) [26]. Here lens distortion correction is included in the affine transformation. However, determining sufficient manual tie points and GCPs represents a significant human resource component in an image processing system. To overcome this problem, georeferencing can also be conducted by photogrammetric approach that consists of four main components: boresight calibration, camera lens distortion correction, obtaining camera position and attitude, and othroimage production. This photogrammetric approach is adopted in this paper.

Typically, the camera, the Global Positioning System (GPS) receiver, and the Inertial Measurement Unit (IMU) are installed individually on an airplane frame. The GPS position is required to shift to the camera position (the perspective center). The boresight calibration is needed to adjust the offset angles between the IMU and camera reference frame axes [27, 28]. The perspective center (X, Y, Z) and three rotation angles (ω, ϕ, κ) are acquired by GPS and IMU measurement, and are associated with the ground coordinate system. The camera position and attitude at the time of imaging can be obtained by interpolating the GPS and IMU data at the imaging time. This procedure is usually completed by dedicated software, which is included in most remote sensing image processing software. An othroimage can be produced based on the optical principle of imaging. According to the camera optical principle, a straight line can be extended from the perspective center of a camera to a pixel on the photo and further to the object of the pixel on ground [24]. This principle is called the collinearity. When the equations of the imaging plane are available, ground coordinates for each pixel can be obtained based on the principle of the collinearity.

2.1.3 Co-registration

The individual orthoimages need to be co-registered so that these images can be mosaicked correctly. The transformation for co-registration can be derived from the tie points generated by feature matching algorithms, such as the scale invariant feature transform (SIFT) algorithm. This algorithm has been used in UAS remote sensing applications [29]. The SIFT algorithm outperformed a number of other local descriptors in evaluations [30]. However, the preliminary experiments of our coastal shoreline UAS images have shown that Autopano Pro [31], one of the widely used commercial versions of the SIFT algorithm, could produce inappropriate tie points for some neighboring images due to no distinctive features being present. Feature-based image registration algorithms also failed in some rangeland and agricultural areas [32, 33]. Our experiments showed that the area-based image similarity could find correct locations at some failure cases by the SIFT algorithm.

This paper adopts global similarity measures with spatial alignment, because these similarity measures monotonically increase with decreasing spatial misalignment [34]. Specifically, the zero-mean cross correlation coefficient is used, as defined as follows:

$$\rho = \frac{\sum_k \left(a_k - \overline{A}\right)\left(b_k - \overline{B}\right)}{\sqrt{\sum_k \left(a_k - \overline{A}\right)^2 \sum_k \left(b_k - \overline{B}\right)^2}} \tag{3}$$

where a_k and b_k are, respectively, the gray-levels of the k pixel in images, or image patches, A and B. $\overline{A}$ and $\overline{B}$ are, respectively, the mean gray levels of A and B. Two questions needed to be addressed before applying the similarity for the co-registration: what is the suitable size of the images? And how big should the search range be?

By our experience, the similarity calculation should use a large array of pixels on the UAS hyper-spatial images to get reliable co-registration. For example, neighboring cotton areas of 1 m by 1 m, even 10 m by 10 m, usually is similar due to the same agricultural crop management. In most Texas coastal areas, natural stable beaches are usually more than 30 m wide [35]. The sand beach actually looks similar everywhere. Empirically, it is desirable for an image patch to be sufficiently large such as 100 m by 100 m for Texas coastal beaches. An image patch of this size usually can encompass several land cover types, for example, fore dune vegetation, beach sand and seawater. This magnitude means an extent of 1000 by 1000 pixels on a hyper-spatial image of 0.1 m pixel size. The optimal match location can be found by traversing the image patch over its neighboring image within a search range (Fig. 2). Hereafter, the image patch is referred to as the match image, and the neighboring image is referred to as the base image.

The search range is determined by both GPS (X, Y, Z) errors and IMU (ω, ϕ, κ) errors. The low cost IMU typically has angular resolution of 0.05°. The 0.05° errors result in positional error up to roughly 10 pixels. The similarity is calculated on neighboring images. The relative locations of two neighboring images are mainly dependent on the angular resolution. Combining with GPS errors, our experience has shown that a suitable search space roughly extends 20 m, which is a search range of 200 pixels. If accurate GPS and IMU are used and the post-processing are carried out well, the search range could be reduced greatly. In contrast, the search range should be increased for the inaccurate GPS and IMU data. The more inaccurate GPS and IMU are, the larger the search range should be. The size of the search range does not depend on the ground scene on the two adjacent images. The cross correlation coefficient is computed directly by using the raw intensity values of corresponding pixels on the match and base images. The UAS photos typically are acquired with high forward and side overlap, for example, 70 % forward and 50 % side. So both the two adjacent images have large portions occupied by a corresponding region. Due to almost the same image acquisition condition, the cross correlation coefficient is an effective similarity measure for the UAS image registration. Our seagrass experiments showed that the coefficient is effective for the UAS images that consist of all water.

The search range for spatial alignment is quite small compared with the image sizes of both the match and base images. For example, if the match image is 1001 by 1001 pixels, and the

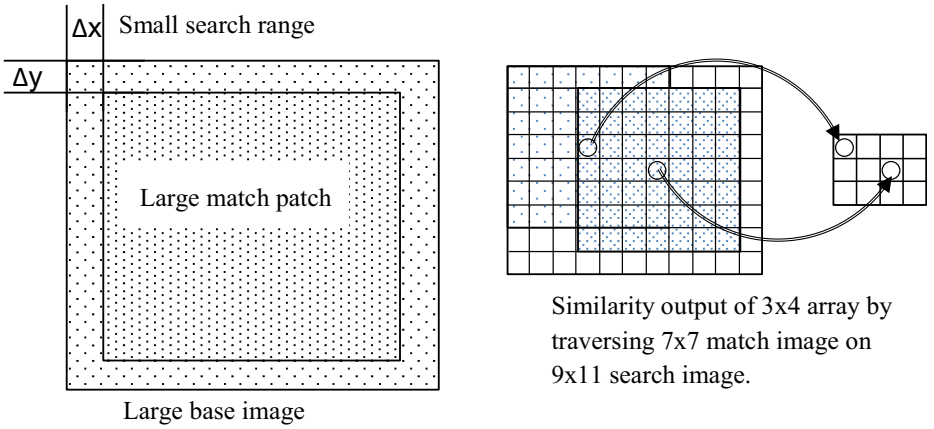

Fig. 2 Similarity between large match image and base image with relative small search range

base image is 1201 by 1201 pixels, the search range is only 200 pixels in two dimensions. While the match image traverses the base image, the center of the match image will form an array of the search extent, for example an array of 200 by 200 pixels. When each element of the search extent array holds a cross-correlation coefficient of the match image and its overlapping region on the base image, the highest similarity is the maximal element of the 200 by 200 pixels.

2.1.4 Mosaicking images

A study area is usually covered by multiple flight lines. In order to reduce error accumulation, the central image is considered as a base image to stitch images along two opposite directions in a flight line. Then single flight line is stitched up, multiple flight lines are mosaicked. After sufficient tie points are identified, the least square method is a widely used method to calculate the optimal parameters for affine transformation, higher-order polynomial transformation, or thin-plate spline. By our experience, an affine transformation is preferred because the UAS orthoimages usually do not needed be bent or curved.

2.2 CUDA architecture and programming model

The Compute Unified Device Architecture (CUDA) introduced by NVIDIA is a general purpose parallel computing architecture. The CUDA computing system consists of a host (namely a traditional central processing unit, CPU) and one or more devices (namely GPU) that are massively parallel processors. A CUDA program consists of one or more phases that are executed on either the CPU or the GPU, depending on the tasks. The phases that exhibit little or no data parallelism are implemented in the CPU code, which is the straight ANSI C code. The phases that exhibit rich amount of data parallelism are implemented in the GPU code, written by so-called kernels. A kernel function is executed by a thread. The GPU threads are the minimum execution units; multiple threads (typically set to a multiple of 16) form a block; and multiple blocks can be composed of a grid, where a grid is the GPU application execution. Normally the total number of threads in a block does not exceed 1024. The blocks are arranged in the form of a grid of 3D array. The values of the 3D grid dimension can range from 1 to 65,535. At any time, each thread executes the same instruction, but operates on different data. Two variables provided by CUDA, blockIdx (the index of a block in a grid) and threadIdx (the index of a thread in a block), are used to localize the data needed processing and to distinguish threads from each other.

In CUDA, the host and devices have separate memory spaces. The GPU has several different memory spaces, and the three main types are: global, constant, and shared. The GPU threads can access multi memory spaces. Each thread has a private memory; each block has a shared memory and each thread in this block can access the shared memory; each thread can access the global memory and the constant memory. Global and constant memories are used for data transfer between host and device. Constant variables are often used for variables that provide input values to kernel function because all threads in a grid see the same version of a constant variable during the entire application execution. Constant variables are stored in the global memory but are cached for more efficient access. The shared memory latency can be roughly 100 times lower than the uncached global memory latency [36]. All blocks in a grid share the global and constant memory. All blocks can run in parallel with each other. All threads in a block are also executed in parallel, and can share data efficiently through the

shared memory and synchronize their execution for coordinating accesses to the shared memory. The profitable strategy for performing computation on the GPUs using this advantage is to divide data in subsets, copy them from global memory to shared memory, achieve shared memory locally in threads, and then copy the results back to global memory.

Different GPU hardware may have different configurations. The CUDA environment takes care of the differences. The transparent scalability let users focus on their applications. Based on the CUDA programming model [36], the basic flow of a standard GPU application is as follows:

a) Declare input and output variables in CPU memory
b) Declare corresponding variables in the GPU global memory using cudaMalloc() function;
c) Transfer input data from GPU global memory to CPU memory using cudaMemcpy() function
d) Run a kernel function by <<<dimGrid, dimBlock>>> instruction;
e) Transfer result data from GPU global memory to CPU memory using cudaMemcpy() function;
f) Free variables in GPU memory using cudaFree() function.

3 UAS hyper-spatial image processing based on CUDA

3.1 The index array approach for the orthoimage generation

This paper presents a GPU approach for UAS hyper-spatial image processing. Figure 3 shows the work flow of the lens distortion correction and the geo-referencing. The models of these two components and the affine transformations are calculated on CPU. The calculation of the four corners and their minimal bounding rectangle (MBR) are also implemented on CPU. This is reasonable because these calculations have little data parallelism. In contrast, the calculation of each pixel is implemented on GPU due to the fact that an image usually consists of a million

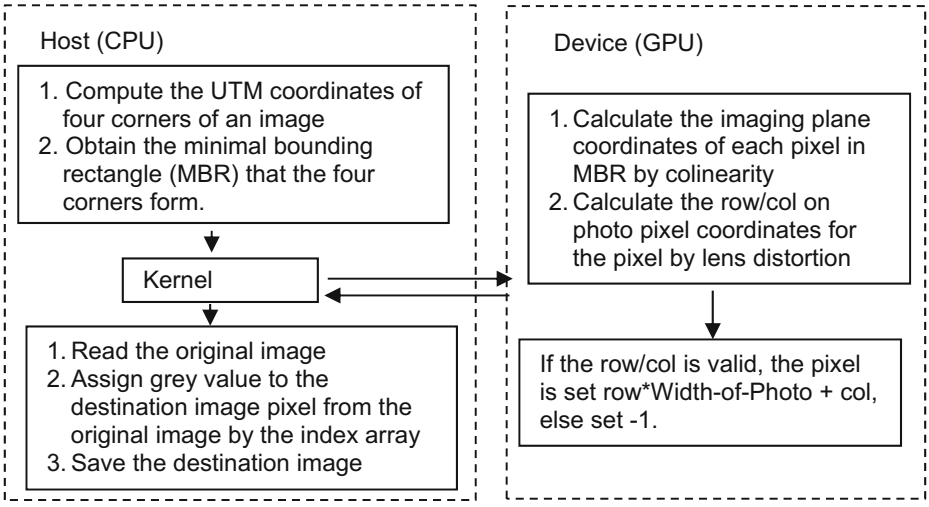

Fig. 3 Work flow of the lens distortion correction and Geo-referencing

pixels and each pixel can be calculated independently in UAS hyper-spatial image processing. Specifically, first the model of lens distortion correction computes the displacement of four corners of a photo to get their correct locations on the imaging plane. Then the four points are projected into the ground coordinate system. The footprint of the destination image will fall in the MBR formed by the four corners. Given a ground pixel size, the coordinates of each pixel in the MBR on the ground coordinate system can be calculated. For any pixel in the MRB, its location on the original photo can be obtained by the collinearity and the lens distortion correction.

Several methods can be used to obtain grey value for the ground pixel of the destination image from the location and its neighbors on the original image. This paper uses the nearest neighbor resampling method due to its simplicity and the capability of preserving original radiometric values. The preservation of the original grey value is very important for UAS remote sensing because UAS remote sensing often uses low radiometric resolution digital cameras. Interpolations of the pixel grey values bring effects of neighboring pixel grey value into the interpolated pixel. This mixing reduces the radiometric resolution further. The nearest neighbor resampling may produce some position errors, especially along linear features. However, for hyper-spatial image of UAS remote sensing the position errors of several pixels do not produce large displacements due to its sub-decimeter pixel size.

An advantage of the nearest neighbor resampling method is that it is not necessary to access grey value of the original image during the lens distortion correction and geo-referencing. The location on the original image (row and column) could be an index, pointing its grey value for the corresponding ground pixel on the destination image. Generating the index array does not require accessing the original image. The destination image can be produced later by assigning its pixel grey value with the index array. We refer to this method as an index array approach.

Another advantage is that calculation of the index array could be parallel executed when the original image is read from disk. This advantage provides additional benefits for remote sensing. As we know that remote sensing images usually are multispectral or hyperspectral images, which have spatially co-registered several spectral bands or hundreds of spectral bands, respectively. This index array approach reduces huge data transfer between CPU and GPU memory spaces.

3.2 Optimizing memory use of the index array approach

The input values of the kernel function include the Universal Transverse Mercator (UTM) coordinates of the left upper corner, height and width of both the original image and its destination image to be produced, and the model coefficients. They are declared as constant variables because each thread uses the same parameters and coefficient.

Besides these parameters and coefficients, all datasets used by the UAS remote sensing image processing are the index array, the original image, and the destination image. The index array is stored in the global memory. In our approach the index array elements do not store a grey value from the original image. They are the location of the nearest neighbor pixel on the original image. The destination image is not produced on GPU side. In this way, both the original image and the destination image are variables in CPU memory. It is not required to transfer the original image from CPU memory to GPU global memory, and to transfer the destination image from GPU memory to CPU memory. Further, because the index array is generated exclusively on GPU, the index array is only transferred from GPU global memory to CPU memory. Due to the independence of the index array elements, the threads do not need to

share data among them. Moreover, since each element of the index array only needs to access global memory once, the strategy of zero-copy host memory for the index array is adopted in this paper.

3.3 Dual tiled similarity computation for image co-registration

This paper presents a dual tiled similarity algorithm for the image co-registration in order to take the advantage of GPU capability. For comparison purpose, the paper also implements two simple algorithms for the similarity calculation: single algorithm and basic algorithm. Assuming that the match image is an array of $N = N_{col} \times N_{row}$ elements and the array of search extent has $M = M_{col} \times M_{row}$ elements. Using Eq. 3, the two means should be obtained before the cross-correlation coefficient can be calculated. The execution time of the three calculations is proportional to the number of elements involved, which is $O(N)$ computational complexity. The reduction algorithm is suitable since all of them derive a single value from an array of values. The match image keeps constant, but it traverses the base image within the search range. In total, the match image has M different overlapping areas on the base image. Total computational complexity is $O(NM)$. The following algorithm analysis focuses on efficiency of the CPU to GPU data transferring, and of accessing the GPU memory variables.

The single algorithm is to transfer the match image and its each overlapping area on the base image from the CPU to the GPU once, to calculate a relationship coefficient within the GPU side, and return the coefficient to the CPU. Thus some pixels of the base image must be repeatedly transferred many times. As shown in Table 1, the single algorithm transfers $N+NM$ data elements from the CPU to the GPU. The total number of global memory accesses is $N(M+1)$ for the means and $2NM$ for the coefficient calculation, where 1 is for the match image itself and M is the total number of the overlapping areas on the base image.

The basic algorithm loads both the match image and the base image entirely into the GPU global memory from the CPU. All coefficients are calculated within the GPU side, the maximal coefficient and its location are returned to the CPU. Although the match and base images are transferred once, some pixels of the base image have to be repeatedly accessed many times. As shown in Table 1, the transferred data has $N_{col} \times N_{row} + (N_{col} + M_{col}) \times (N_{row} + M_{row})$ data elements. Without loss of generality, we can assume $M_{col} = N_{col}/4$ and $M_{row} = N_{row}/4$. The basic algorithm transfers $2.5N + M$ data elements. The total number of global memory accesses that the basic algorithm requires is the same as the single algorithm.

Table 1 The requirement of transferring data to GPU and of accessing GPU global memory and computational complexity

Algorithms	Data transferring to GPU	GPU global data accessing	Computational complexity
Single	N+NM	N+NM for means M×2 N for coefficients	$O(NM)$
Basic	2.5×N+M	N+NM for means M×2 N for coefficients	$O(NM)$
Dual tilted	$N \times \left(2 + \frac{M_{row}+M_{col}}{P_{width}} + \frac{M}{P_{width}{}^2}\right)$	N for mean of the match image $18N \times \left(1 + \frac{M_{col}}{P_{width}}\right) \times \left(1 + \frac{M_{row}}{P_{width}}\right)$ for the mean and the coefficient of base image	$O(NM)$

Among the three types of GPU memory spaces, global memory is large (typically gigabyte size) but slow whereas the constant memory and the shared memory are small (typically kilobyte size) but fast. Both the constant memory and the shared memory have no enough room to entirely hold the match or base images, which usually has a megabyte size. The dual tiled algorithm efficiently uses their size and access speed (Fig. 4). After loading the match and base images into the global memory, this algorithm uses dual partition strategies, namely partitions on both the match and base images. The first partition divides the match image into small pieces so that each piece can fit into the constant memories. The second partition organizes the base image into tiles. An important criterion of the tile size is that the kernel computation on these tiles can be done independently of each other. The partitions should be careful not to exceed the capacity of these memories for kernel execution. It is worth noting that the size of both the constant and the shared memories can vary from device to device. A device property query, cudaGetDeviceProperties(), can obtain their amount available on a device.

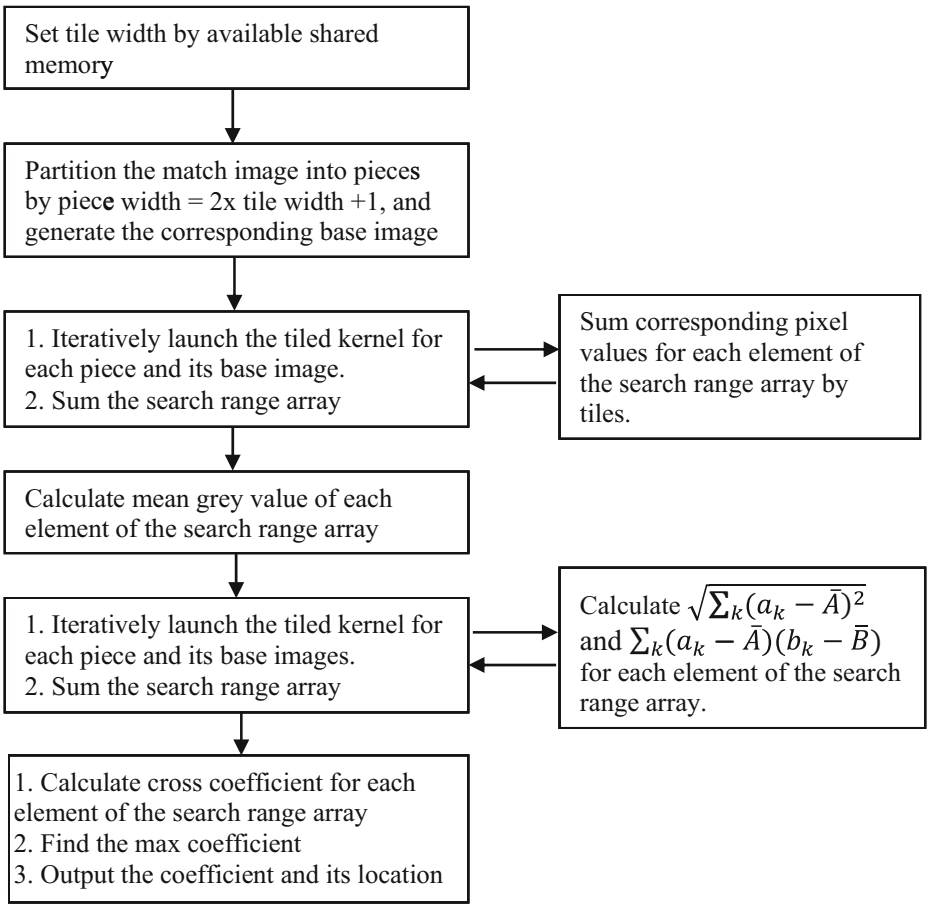

Fig. 4 Workflow of the dual tiled image match algorithm

The three components of Eq. 3, namely $(a_k-\overline{A})(b_k-\overline{B})$, $(a_k-\overline{A})^2$, and $(b_k-\overline{B})^2$, are the sum of all pixel pairs on the match image and its overlapping region on the base image. This property allows the three components to be calculated by regions. As discussed, all possible cross-coefficients form an array of $M_{\text{col}} \times M_{\text{row}}$ elements. The three components may be produced by summing corresponding components of all match image pieces when each single piece has an array of $M_{\text{col}} \times M_{\text{row}}$ pixels to hold its three quantities. Each piece of the match image is loaded into the constant memory iteratively. With the tiles, the three components of each single piece are calculated with reduced global memory accesses comparing with the above basic algorithm. To obtain high efficiency, the size of the match image pieces should be bigger than the tiles of the base image. The three quantities calculated for a tile needs elements from neighbor tiles of the base image. The elements involved in multiple tiles and loaded by multiple blocks are commonly referred to as *halo elements*. The sizes of the piece and the tile determine number of the halo elements and which elements are the halo elements. For simplicity, width of the match image piece can be set equal to one plus double width of the tile, namely $P_{\text{width}} = 2 \times T_{\text{width}} + 1$. Each element in a tile has eight halo elements on the corresponding location in its eight neighboring tiles (Fig. 5). Computing the three quantities of all elements in a tile needs to load nine tiles into the shared memory. For the illustration purpose, in Fig. 5, the match image piece is an array of 5 by 5 pixels and the tiles have 2 by 2 pixels. Comparing with the basic algorithms, the ratio of memory accesses for once calculation of the three quantities is:

$$R_{\text{access}} = \left({T_{\text{width}}}^2 \times 9\right) \Big/ \left({P_{\text{width}}}^2 \times {T_{\text{width}}}^2\right) \tag{4}$$

If and only if $T_{\text{width}} \geq 2$, $R_{\text{access}} > 1$. The ratio of memory access reduction is approximately proportional to the tile size, namely $T_{\text{width}} \times T_{\text{width}}$.

The match image is partitioned into ceil($N_{\text{col}}/P_{\text{width}}$) by ceil($N_{\text{row}}/P_{\text{width}}$) pieces that has $P_{\text{width}} \times P_{\text{width}}$ elements. Each piece of the match image has its own base image of $(P_{\text{width}} + M_{\text{col}}) \times (P_{\text{width}} + M_{\text{row}})$ elements, which is partitioned into tiles of ceil($(P_{\text{width}} + M_{\text{col}})/T_{\text{width}}$) by ceil($(P_{\text{width}} + M_{\text{row}})/T_{\text{width}}$) elements. Each piece of the match image requires transferring

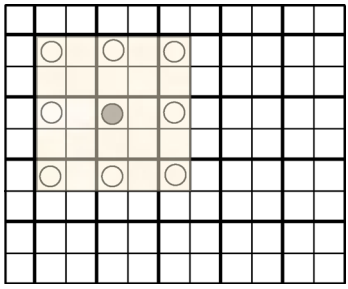

Fig. 5 The *halo* elements on the similarity computation

An element (solid circle) and its *halo elements* (empty circles) for the 2D similarity kernel

$P_{\text{width}} \times P_{\text{width}}$ and $(P_{\text{width}} + M_{\text{col}}) \times (P_{\text{width}} + M_{\text{row}})$ data elements from the CPU to the GPU. Total number of transferring data from the CPU to the GPU is:

$$T_{\text{transfer}} = \text{ceil}\left(\frac{N_{\text{col}}}{P_{\text{width}}}\right) \times \text{ceil}\left(\frac{N_{\text{row}}}{P_{\text{width}}}\right) \times \left(P_{\text{width}} \times P_{\text{width}} + (P_{\text{width}} + M_{\text{col}}) \times (P_{\text{width}} + M_{\text{row}})\right) \quad (5)$$

Without loss of generality, we can assume all divisions leading to an integer. Equation 6 can be rewritten as:

$$T_{\text{transfer}} = N \times \left(2 + \frac{M_{\text{row}} + M_{\text{col}}}{P_{\text{width}}} + \frac{M}{{P_{\text{width}}}^2}\right) \quad (6)$$

For a single piece of the match image, each tile requires ${T_{\text{width}}}^2 \times 9$ global memory accesses for calculating the mean or the coefficient. That is, total number of memory accesses for the whole match image is:

$$T_{\text{access}} = \text{ceil}\left(\frac{N_{\text{col}}}{P_{\text{width}}}\right) \times \text{ceil}\left(\frac{N_{\text{row}}}{P_{\text{width}}}\right) \times \text{ceil}\left(\frac{P_{\text{width}} + M_{\text{col}}}{T_{\text{width}}}\right) \times \text{ceil}\left(\frac{P_{\text{width}} + M_{\text{row}}}{T_{\text{width}}}\right) \times \left({T_{\text{width}}}^2 \times 9\right) \quad (7)$$

Without loss of generality, we can assume all divisions leading to an integer. Equation 5 can be rewritten as:

$$T_{\text{access}} = 9\text{N} \times \left(1 + \frac{M_{\text{col}}}{P_{\text{width}}}\right) \times \left(1 + \frac{M_{\text{row}}}{P_{\text{width}}}\right) \quad (8)$$

Due to insufficient amount of on-chip memory, P_{width} usually is not big. It is worth noting that one T_{access} is for the mean, and another for the cross-relationship coefficient itself. The total global memory accesses of the tiled algorithm are:

$$T_{\text{access}} = \text{N} + 18\text{N} \times \left(1 + \frac{M_{\text{col}}}{P_{\text{width}}}\right) \times \left(1 + \frac{M_{\text{row}}}{P_{\text{width}}}\right) \quad (9)$$

4 Experiments and discussions

4.1 The index array approach

A total of eight experiments were designed and conducted to process two UAS datasets of South Padre island shorelines (Table 2). The computation consists of lens distortion correction and geo-referencing. The eight experiments consist of four implementations for the two different data sets. The four implementations include CPU implementation, GPU implementation with global memory, and two GPU implementations with two different zero-copy host memory settings. The data sets are small frames of 1920x1600 pixels with unsigned 8 bit integer and large frames of 7378x4924 pixels with unsigned 16 bit integer; the size of output images is 6810x4790 and 1586x1178, respectively.

Table 2 Speedup of the eight computations (unit: milliseconds)

Data	Implementation	Generating index array	Copying index to CPU	Reading single band	Assigning grey value	Writing single band	total
Large frames: Input: 7378 × 4924 Output: 6810 × 4790 unsigned 16 bit integer	Zero-copy 2	660	0	3323	2331	5014	11,328
	Zero-copy 1	654	0	3092	124	4980	8850
	GPU global	660	34	3201	123	5225	9243
	CPU	7498	0	3186	122	4995	15,801
Small frames: Input: 1920 × 1600 Output: 1586 × 1176 unsigned 8 bit integer	Zero-copy 2	36	0	2362	7	574	2979
	Zero-copy 1	37	0	2318	7	524	2886
	GPU global	37	2	2212	9	479	2739
	CPU	610	0	2338	10	497	3455

The computer that was used was a Dell Precision T3600 (CPU: 3.60GHz Intel Xeon E5-1620, RAM: 32 GB), Windows 7 Professional 64-bit operating system, and Visual Studio 2010 development environment. GPU device was Quadro 600, compute capability 2.1, 1 GB global memory, 64 KB constant memory, and CUDA 6.0 runtime.

As shown in Table 2, the calculation time (the column of Generating index array) is short, but the image file reading and writing time is long. It is not surprising that the reading and writing operation is the bottleneck of the efficiency of the entire processing. Speedup analysis did not include the reading and writing operations. The speedup is 16 times on the small frames and 11 times on the large frames. For the CPU implementation all computations were done in CPU. For the GPU global implementation, the index array resides in GPU global memory. In addition to the kernel execution, transferring data between GPU and CPU memory costs time. The time of transferring the index array from GPU to CPU was 2 ms for the small frames and 34 ms for the large frames.

Since each element of the index array is accessed exactly once, we expected a performance enhancement when using zero-copy memory, in which we access CPU data directly from GPU. The zero-copy memory has two different settings by two flags of cudaHostAlloc() function, which allocates memory on CPU. The two flags are cudaHostAllocMapped and cudaHostAllocWriteCombined. The flag cudaHostAllocMapped shows that this buffer is accessed from GPU. The flag cudaHostAllocWriteCombined indicates to allocate the memory as write-combined (WC). WC memory is a good option for buffers that is written by the CPU and read by the GPU, but it cannot be read efficiently by most CPUs. Here Zero-copy 1 was to allocate memory on CPU with the flag cudaHostAllocMapped only while Zero-copy 2 was

Table 3 Speedup of the similarity computations

Implementation	Total elapsed time (unit: milliseconds)	Speedup			
		CPU	Single	Basic	Dual tiled
CPU	1,427,426	1			
Single	440,539	3.24	1		
Basic	342,101	4.17	1.29	1	
Dual tiled	117,151	12.18	2.92	3.76	1

with both the two flags. As shown in Table 2, for the large frames, Zero-copy 2 spent almost 20 times longer to assign grey values to the destination image than Zero-copy 1, but the time difference was ignorable among CPU, GPU global, and Zero-copy 1 implementations. In contrast, for the small frames, the time of assigning grey values was roughly the same for both CPU and all GPU implementations.

4.2 The dual tiled similarity computation

This experiment used 32 pair of the 0.1 m UAS image that has 7378×4924 pixels with unsigned 16 bit integer. The average size of the match image is 1121×1121 pixels, namely 112.1 m by 112.1 m patch. The search range is from −10 m to 10 m from the original location of the match base on the base image. In other words, the array of the search extent is

Fig. 6 Study site. The star in the right upper figure represents its location in Texas. The red rectangle in the left figure is its position on Padre Island (source: Google Maps). The yellow rectangle in the right lower figure represents its range on the beach (source: Google Maps)

200 × 200 pixels. The average cross-correlation coefficient is 0.95, and the minimal is 0.89. Table 3 shows the speedup for the four implementations: the CPU, the single, the basic, and the dual tiled algorithms. All GPU implementations generated significant time reductions. The dual tiled algorithm obtained speedup of 12 times compared with the CPU computation.

4.3 An example on Texas coastal environment

The study site is a beach on South Padre island, which is a barrier island in the U.S. state of Texas (Fig. 6). The UAS images were acquired on March 4 and June 25, 2014. The unmanned aerial vehicle was American Aerospace RS-16 owned by the TAMU-CC UAS program. The March exercise used Nikon D800 camera with 50 mm of focal length and 0.0049 mm of CCD pixel size. The flight height was 870 m above the ground. The June exercise used the camera of 14.85 mm focal length and 0.0025 mm pixel size. Its flight height was 300 m. Figure 7 shows two raw photos and their geo-referenced images. Both geo-referenced images are with 10 cm pixel size. The large frame image has much greater footprints than the small frame image. The shoreline indicators, such as seaward dune vegetation line and instantaneous water line, are discernible visibly in the UAS coastal imagery. Figure 8 displays two mosaic images generated with 23 large frame photos and 18 small frame photos, respectively. The left picture is the two overlapped mosaic images by WGS 84 UTM 14 N zone. The right picture is the enlarged mosaic image of the small frame photos. As the large frame photos were acquired during March, no new vegetation and algae accumulation are on beach. On the June images the opportunistic plants covered the landward portion of the dry beach; and huge volumes of

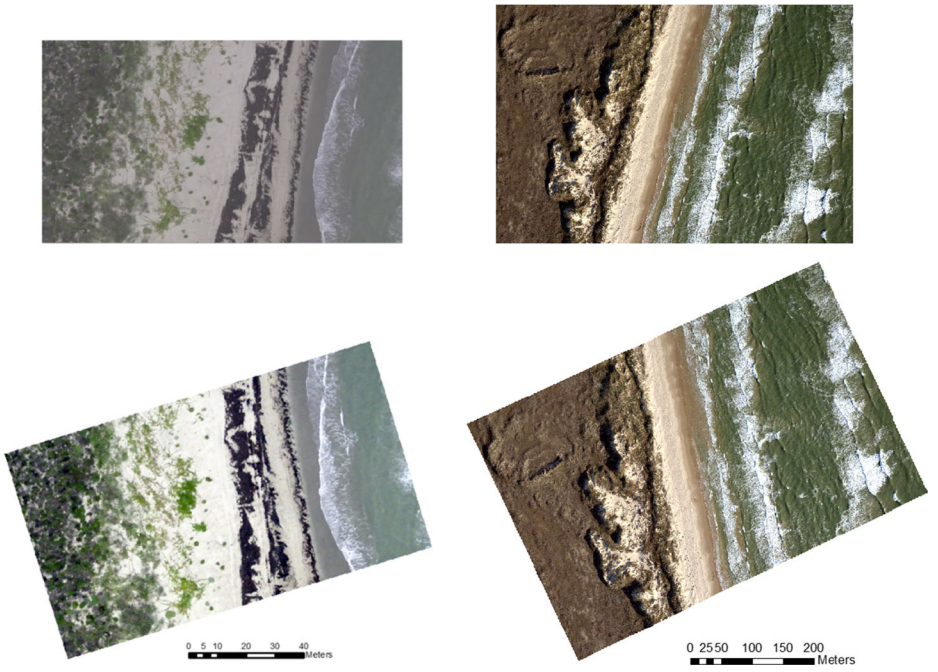

Fig. 7 Raw photos (unscaled) and geo-referenced images. The upper left picture is a photo of the small frame 1920 × 1600. The upper right one is a photo of large frame 7378 × 4924. The lower pictures are their geo-referenced images shown with different scales

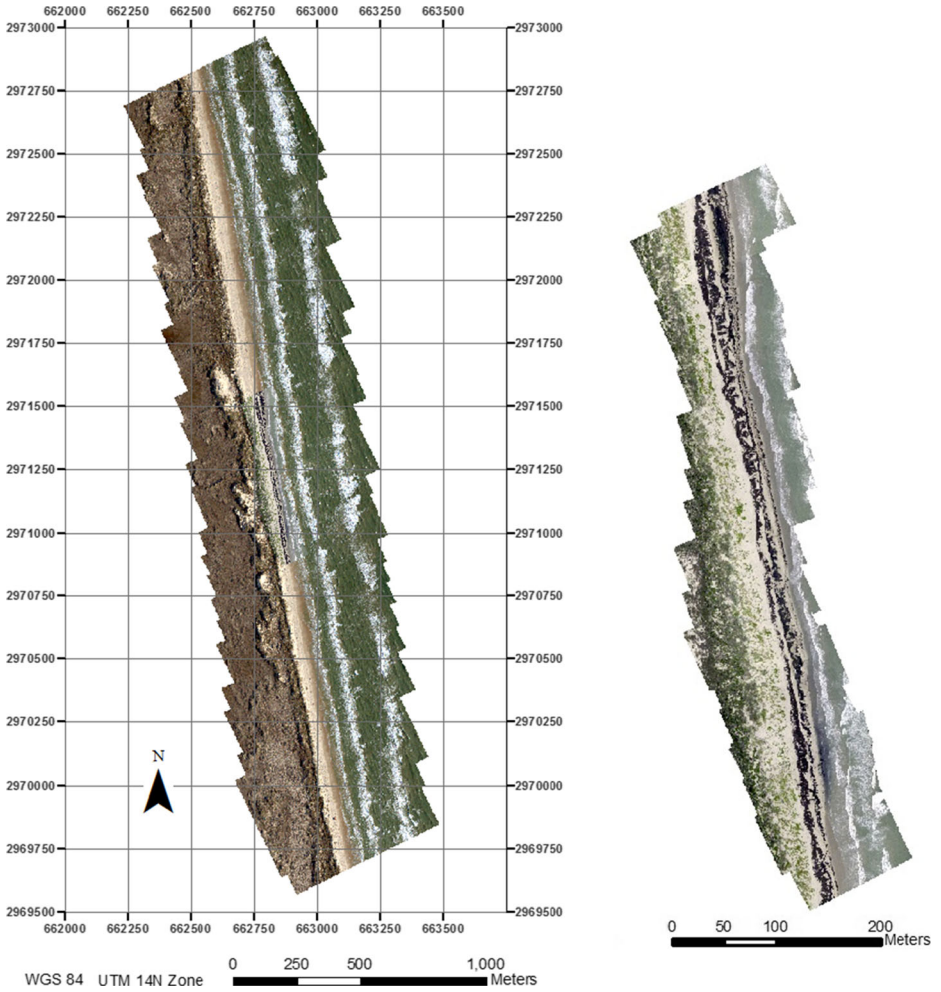

Fig. 8 Two mosaic images generated with 23 large frame photos and 18 small frame photos, respectively, in WGS 84 UTM 14 N zone. The left figure is the two overlapped mosaic images. The right figure is the enlarged mosaic image of the small frame photos

the algae were onto the beach in rows. This example shows that UAS remote sensing provides a new tool for monitoring coastal dynamic changes.

5 Conclusion

This paper presents an index array approach of processing hyper-spatial images of UAS remote sensing. The experiments of the lens distortion correction and geo-referencing algorithm showed a significant speedup. In the case of not considering image loading and saving, speedup was at least 11 times better than that would have been with the traditional CPU approach. The dual tiled algorithm can obtain 12 times speedup relative to the CPU similarity computation. The experiments showed that these two approaches have improved efficiency of

UAS remote sensing image processing. The GPU computation is helpful to address big data volume and demand time limit, which are the two main challenges that the UAS remote sensing faces.

Acknowledgments This work was supported partly by the NSF grant MRI: Acquisition of a High Performance Computing Cluster to Support Multidisciplinary Big Data Analysis and Modeling (#1429518).

References

1. FAA, Unmanned Aircraft Systems (UAS) - Regulations & Policies, 2011, http://www.faa.gov/about/initiatives/uas/reg/
2. Christophe E, Michel J, Inglada J (2011) Remote sensing processing: from multicore to GPU. IEEE J Sel Top Appl Earth Obs Remote Sens 4(3):643–652
3. Lee CA, Gasster SD, Plaza A, Chang C, Huang B (2011) Recent developments in high performance computing for remote sensing: a review. IEEE J Sel Top Appl Earth Obs Remote Sens 4(3):508–527
4. Setoain J, Prieto M, Tenllado C, Plaza A, Tirado F (2007) Parallel morphological endmember extraction using commodity graphics hardware. IEEE Geosci Remote Sens Lett 4(3):441–445
5. Yang H, Du Q, Chen G (2011) Unsupervised hyperspectral band selection using graphics processing units. IEEE J Sel Top Appl Earth Obs Remote Sens 4(3):660–668
6. Luo Y, Guo K, Zhao S, Tao Z, Wang M (2012) Feature extraction of hyperspectral remote sensing in parallel computing research based on GPU. In Proceedings of the 4th GEOBIA, pp. 381–384, May, Rio de Janeiro, Brazil
7. Trigueros-Espinosa B, Velez-Reyes M, Santiago NG, Rosario-Torres S (2012) Evaluation of the graphics processing unit architecture for the implementation of target detection algorithms for hyperspectral imagery. Journal of Applied Remote Sensing, Vol.6, No.1, 061506 (Jun 21). doi: 10.1117/1.JRS.6.061506
8. Gonzalez C, Sanchez S, Paz A, Resano J, Mozos D, Plaza A (2013) Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing. Integr VLSI J 46:89–103
9. Qu H, Zhang J, Lin Z, Chen H (2013) Parallel Acceleration of SAM Algorithm and Performance Analysis. IEEE J Sel Top Appl Earth Obs Remote Sens 6(3):1172–1178
10. Molero JM, Garzon EM, Garcia I, Quintana-Orti ES, Plaza A (2014) Efficient implementation of hyperspectral anomaly detection techniques on GPUs and multicore processors. IEEE J Sel Top Appl Earth Obs Remote Sens 7(6):2256–2266
11. Nascimento JMP, Bioucas-Dias JM, Alves JMR, Silva V, Plaza A (2014) Parallel hyperspectral unmixing on GPUs. IEEE Geosci Remote Sens Lett 11(3):666–670
12. Wu X, Huang B, Plaza A, Li Y, Wu C (2014) Real-time implementation of the pixel purity index algorithm for endmember identification on GPUs. IEEE Geosci Remote Sens Lett 11(5):955–959
13. Dai C, Yang J (2011) Research on orthorectification of remote sensing images using GPU-CPU cooperative processing. In Proceedings of IGARSS 2011. July, Vancouver, Canada
14. Reguera-Salgado J, Calvino-Cancela M, Martin-Herrero J (2012) GPU geocorrection for airborne pushbroom imagers. IEEE Trans Geosci Remote Sens 50(11):4409–4419
15. Lemoine G, Giovalli M (2013) Geo-correction of high-resolution imagery using fast template matching on a GPU in emergency mapping contexts. Remote Sens 5:4488–4502. doi:10.3390/rs5094488
16. Zhang W, Li Y, Li D, Teng C, Liu J (2014) Distortion correction algorithm for UAV remote sensing image based on CUDA. 35th International Symposium on Remote Sensing of Environment (ISRSE35), IOP Conference Series: Earth and Environmental Science, vol.17
17. Fang L, Wang M, Li D, Pan J (2014) CPU/GPU near real-time preprocessing for ZY-3 satellite images: relative radiometric correction, MTF compensation, and geocorrection. ISPRS J Photogramm Remote Sens 87:229–240
18. Lei Z, Wang M, Li D, Lei TL (2014) Stream model-based orthorectification in a GPU cluster environment. IEEE Geosci Remote Sens Lett 11(12):2115–2119
19. Su X, Wu J, Huang B, Wu Z (2013) GPU-accelerated computation for electromagnetic scattering of a double-layer vegetation model. IEEE J Sel Top Appl Earth Obs Remote Sens 6(4):1799–1806
20. Coleman DM, Feldman DR (2013) Porting existing radiation code for GPU acceleration. IEEE J Sel Top Appl Earth Obs Remote Sens 6(6):2486–2491
21. Fulkerson B, Soatto S (2012) Really quick shift: Image segmentation on a GPU. Trends and Topics in Computer Vision. Springer, Berlin, pp 350–358

22. Happ PN, Feitosa RQ, Bentes C, Farias R (2013) A region-growing segmentation algorithm for GPUs. IEEE Geosci Remote Sens Lett 10(6):1612–1616
23. Bernabe S, Plaza A, Marpu PR, Benediktsson JA (2012) A new parallel tool for classification of remotely sensed imagery. Comput Geosci 46:208–218
24. Wolf P, DeWitt B (2000) Elements of photogrammetry with applications in GIS (3rd Edition). McGraw-Hill, New York
25. (2013) Intergraph, ERDAS Field Guide
26. Lillesand T, Kiefer RW, Chipman J (2007) Remote sensing and image interpretation (6th Edition). Wiley, New York
27. Maune DF (2007) Digital elevation model technologies and applications: the DEM users manual. The American Society for Photogrammetry and Remote Sensing
28. Shan J, Toth CK (2008) Topographic laser ranging and scanning: principles and processing. CRC
29. Turner D, Lucieer A, Malenovsky Z, King DH, Robinson SA (2014) Spatial co-registration of ultra-high resolution visible, multispectral and thermal images acquired with a micro-UAV over antarctic moss beds. Remote Sens 6(5):4003–4024. doi:10.3390/rs6054003
30. Mikolajczyk K, Schmid C (2005) A performance evaluation of local descriptors. IEEE Trans Pattern Anal Mach Intell 27(10):1615–1630
31. (2014) Kolor Autopano, Version 1.4.0. Autopano website http://www.autopano.net
32. Laliberte AS, Herrick JE, Rango A, Winters C (2010) Acquisition, orthorectification, and object-based classification of Unmanned Aerial Vehicle (UAV) imagery for rangeland monitoring. Photogramm Eng Remote Sens 76(6):661–672
33. Du Q, Raksuntorn N, Orduyilmaz A, Bruce LM (2008) Automatic registration and mosaicking for airborne multispectral image sequences. Photogramm Eng Remote Sens 74(2):169–181
34. Mitchell HB (2010) Image fusion theories, Techniques and applications. Springer, New York
35. Morton RA, Peterson RL, South Texas Coastal Classification Maps - Mansfield Channel to the Rio Grande, USGS Open File Report 2006–1133. http://pubs.usgs.gov/of/2006/1133/mapintro.html Accessed 21 Oct 2015
36. Kirk DB, Hwu WW (2010) Programming massively parallel processors, a hands-on approach. Elsevier, New York

Lihong Su is an Associate Research Scientist at the Harte Research Institute for Gulf of Mexico Studies (HRI) at Texas A&M University – Corpus Christi. He earned a bachelor's degree in Mathematics from Xinjiang University, a Master's degree in Geography Information System, and a PhD in remote sensing from Chinese Academy of Sciences. He is interested in spatial data handling, remote sensing applications and computer simulation in GIS and remote sensing.

Yuxia Huang received the Ph.D degree in Geographic Information Science from the University at Buffalo, State University of New York in 2009. She is currently with the School of Engineering and Computing Science, Texas A&M University – Corpus Christi, as an associate Professor. She is also a research scientist in the Conrad Blucher Institute, Texas A&M University – Corpus Christi. Her research focuses on semantic integration for spatial information and ontology construction, and GIS applications in health.

James Gibeaut is the Endowed Chair for Coastal and Marine Geospatial Sciences at the Harte Research Institute for Gulf of Mexico Studies (HRI) and Associate Professor of Geology at Texas A&M University – Corpus Christi. He earned a B.S. in geology from Ohio State University, a M.S. in coastal geology from the University of Rhode Island, and a Ph.D. in Marine Science from the University of South Florida. He is a coastal geologist who uses optical, radar, and lidar remote sensing, GIS, and field surveys to measure and understand coastal change. He has studied shorelines in a variety of locations including Rhode Island, Florida, Texas, Alaska, Honduras, Venezuela, Brazil, and Saudi Arabia. Currently, his main research focus is modeling the effects of relative sea-level rise and storms on coastal systems and projecting future change. His Coastal and Marine Geospatial Lab at HRI is also developing web applications and scientific data repositories for the dissemination of research results.

Longzhuang Li obtained his Bachelor and Master degree in 1992 and 1995, respectively at Northwestern Polytechinc University, China, and Ph.D degree in 2002 at University of Missouri-Columbia. Now he is an associate professor at Texas A&M University-Corpus Christi. His research focuses on data integration, data mining and big data processing, and has been supported by National Science Foundation and Air Force Research Lab.