remote sensing

MDPI

Article

A GPU-Based Integration Method from Raster Data to a Hexagonal Discrete Global Grid

Senyuan Zheng [1,2,3], Liangchen Zhou [1,2,3,*], Chengshuai Lu [1,2,3] and Guonian Lv [1,2,3]

[1] Key Laboratory of Virtual Geographic Environment, Ministry of Education, Nanjing Normal University, Nanjing 210023, China; 211301033@njnu.edu.cn (S.Z.); 201345040@njnu.edu.cn (C.L.); gnlu@njnu.edu.cn (G.L.)
[2] State Key Laboratory Cultivation Base of Geographical Environment Evolution (Jiangsu Province), Nanjing 210023, China
[3] Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing 210023, China
* Correspondence: zhoulch@njnu.edu.cn; Tel.: +86-13913968090

Abstract: This paper proposes an algorithm for the conversion of raster data to hexagonal DGGSs in the GPU by redevising the encoding and decoding mechanisms. The researchers first designed a data structure based on rhombic tiles to convert the hexagonal DGGS to a texture format acceptable for GPUs, thus avoiding the irregularity of the hexagonal DGGS. Then, the encoding and decoding methods of the tile data based on space-filling curves were designed, respectively, so as to reduce the amount of data transmission from the CPU to the GPU. Finally, the researchers improved the algorithmic efficiency through thread design. To validate the above design, raster integration experiments were conducted based on the global Aster 30 m digital elevation dataDEM, and the experimental results showed that the raster integration accuracy of this algorithms was around 1 m, while its efficiency could be improved to more than 600 times that of the algorithm for integrating the raster data to the hexagonal DGGS data, executed in the CPU. Therefore, the researchers believe that this study will provide a feasible method for the efficient and stable integration of massive raster data based on a hexagonal grid, which may well support the organization of massive raster data in the field of GIS.

Keywords: DGGS; raster data; GPU; data resampling; encoding and decoding; thread design

Citation: Zheng, S.; Zhou, L.; Lu, C.; Lv, G. A GPU-Based Integration Method from Raster Data to a Hexagonal Discrete Global Grid. *Remote Sens.* **2024**, *16*, 2022. https://doi.org/10.3390/rs16112022

Academic Editors: Marco Painho and Yongze Song

Received: 1 March 2024
Revised: 7 May 2024
Accepted: 3 June 2024
Published: 4 June 2024

1. Introduction

Discrete global grid systems (DGGSs) are frameworks for a digital Earth that enable the division of the Earth's surface into seamless non-overlapping multilevel collections of regional cells for fitting the Earth's surface at different resolutions [1,2]. The raster data model is a data model that selectively displays, locates, and stores real-world environments in a regular grid system, suitable for expressing continuous spatial data. However, with the expansion of GIS application areas, the volume of raster data is also growing larger and larger, and how to efficiently organize and manage massive raster data has become a challenge for GIS researchers. Previous studies have found that DGGSs are more suitable for large-scale applications than traditional local geospatial data organization models and can be structured to support efficient multi-resolution geospatial data processing [3]. Furthermore, rasters may result in an imbalanced generalization in different directions. Compared to the rasters, a hexagon has a consistent connectivity and isotropic neighborhoods, and the results of the hexagon-based method are more balanced in all neighborhood directions, as hexagons match better with the original polygons and have smoother simplified boundaries [4]. In view of this, it would be desirable to devise an efficient algorithm to manage massive raster data with DGGS data organization and scheduling, thus providing a solution to the massive raster data organization and management problems in the current GIS field.

In recent years, several scholars have conducted research on DGGS-based raster data integration. For example, Rui Wang [5] developed an indexing method based on icosahedral mixed-aperture hexagonal grids and implemented a raster integration algorithm design based on this. Mingke Li [6] investigated multi-resolution raster integration in a hexagonal DGGS environment and manipulated the data after integration with descriptive statistics, terrain parameters, and terrain metrics. Andrew Rawson [7] used grid centroids to resample the raster surface to achieve raster integration. However, these research results are currently implemented based on CPUs and, thus, do not allow for efficient data integration when confronted with massive multi-resolution raster data. The root of this problem is that the integration of massive raster data is inherently a computationally intensive application, and the computational performance of CPUs is not sufficient to support such a large amount of computation. When the resolution is high, due to the rapid growth of computational complexity, the computational time and efficiency of CPUs are often inadequate to meet the demands of real-time operation, resulting in the inability to obtain the integration results within a reasonable time.

It is also known that parallel processing is a commonly adopted computational strategy for big data processing and analysis [8], and it is a good idea to use parallel computing techniques to solve the efficiency problems posed by computationally intensive applications. There are various ways to utilize parallelism to improve the performance of data-intensive applications, among which GPUs are the dominant platform [9]. Many GIS algorithms can benefit from the GPU-based parallel computing of geospatial data, such as map matching, view analysis, spatial connectivity, spatial overlay [10], etc. Therefore, many scholars have also conducted research on GPU-based raster computation and analysis. However, a raster is a regular grid structure with the characteristics of a clear spatial relationship and a simple calculation mechanism, and it has a high adaptability to the threaded grid structure of the GPU, while a global discrete grid is not regularly arranged and does not have the nature of a regular grid, so it is difficult to directly apply research results in the form of regular grids to global discrete grids.

Proceeding from previous studies, the authors of this paper devised an algorithm, the core idea of which was to first organize hexagonal DGGS data with a rhombic tile structure, so as to convert them into data with a GPU-acceptable texture structure. Then, the corresponding encoding and decoding methods were designed. On this basis, the acceleration of the algorithm was realized by parallel computing of the GPU. For this purpose, the thread structure between the GPU and the CPU was devised. The rest of this paper is organized as follows: Section 2 provides a review of related studies and an overview of the methodology of this paper. Section 3 presents the basic ideas and the implementation flow design of the GPU-based hexagonal grid integration algorithm. Section 4 discusses and implements the basic ideas and key issues based on the implementation flow proposed in Section 2. The algorithmic flow proposed in this paper is experimentally verified through a case study in Section 5. The major ideas and findings of this study are finally discussed and summarized in Sections 6 and 7.

2. Related Work

The efficiency of GPU-based integration algorithms from raster data to DGGSs is mainly affected by three parameters. The first comprises the encoding methods of DGGSs, the second one consists of the specific algorithm of raster data integration based on DGGSs, and the third one is the application of GPU technology. Therefore, this paper summarizes, compares, and analyzes related research in terms of the above three aspects.

2.1. Encoding Methods of DGGSs

In DGGSs, the cell indexing method usually serves the purpose of providing access to the data. For each cell present in a DGGS, the indexing method assigns an index that uniquely identifies the cell [11]. For DGGS-based raster data integration algorithms, an efficient DGGS index can enhance the efficiency of the algorithm.

In recent years, many scholars have studied the indexing method of hexagonal cells, such as Tong [12], who improved the generalized balanced ternary system (GBT) of satellite remote sensing data, designed hexagonal DGGS hierarchical encoding, which allocates low-level hexagonal cells to multiple high-level hexagonal cells, and realized the seamless representation and processing of global multi-resolution remote sensing data. Vince [13] proposed an indexing method for hexagonal grids based on the hierarchical index structure PYXIS, but the method was designed for hexagonal grids with aperture 3, which required the hierarchical indexing of odd and even levels, leading to complex indexing calculations. White [14] used Morton space-filling curves to index grid data based on polyhedral rhombic surfaces, which improved the retrieval and computational efficiency of grid data, but it did not further subdivide the rhombic surfaces and, thus, did not have a hierarchical structure to support parent–child relationship search, distance search, and range search for the grid data stored inside the rhombus. Tong [15] proposed the hexagonal quaternary balanced structure (HQBS) and established hierarchical relationships among different levels of hexagonal grids on the icosahedral triangle. However, operations frequently roll back when code normalization fails, resulting in a low efficiency [16]. A Mahdavi-Amiri [17] presented a general hierarchical indexing mechanism for hexagonal cells resulting from the refinement of a triangular spherical polyhedral. However, this encoding method is limited to the A3H DGGS, and, because the A3H DGGS rotates between different levels, there are inevitably a lot of logical judgments when decoding the codes. Ali Mahdavi-Amiri [18] implemented general mapping between DGGS grids and exercised a possible solution for extracting a rhombic tile structure from an hexagonal grid by splitting hexagons. Although this research successfully converted an hexagonal DGGSs to a rhombic texture, it did not use a GPU-friendly encoding method. The main coding method used in the current research is the one proposed in the literature [18], which requires additional vertices to be recorded at the time of coding and needs to be decoded twice while performing raster integration operations in the GPU, i.e., by first decoding the rhombic code into a hexagonal code and then decoding the hexagonal code into geographic coordinates. Coding and decoding methods are the cores of DGGSs; therefore, the shortcomings of coding and decoding methods inevitably have a great impact on the efficiency of the conversion from raster data to DGGSs in the GPU.

It can be seen from the above results that the encoding research on hexagonal DGGSs is quite mature, but the above encoding schemes often entail a lot of condition judgments or need to use a pre-calculated code addition table when conducting hexagonal cell search, neighborhood query, parent–child relationship query, and other operations. For example, PYXIS needs to first judge the parity of the DGGS level when conducting a topology query, and HQBS needs to match the codes to the code addition table when conducting a topology query.

In computers, CPUs are good at process control and logic processing and are able to handle irregular data structures, while GPUs are good at the parallel computation of data with regular data structures and not good at computing irregular data, which is due to the architectural differences between CPUs and GPUs, as shown in Figure 1. The CPU is composed of about 25% ALUs (Arithmetic Units), 25% Control (Logic Control Units), and 50% Cache (Cache Units). It is adept at handling small and complex tasks: a typical example of this is the intersection operation between rays and some irregular objects in the same space. The GPU, on the other hand, is composed of 90% ALUs, 5% Control, and 5% Cache. Compared to the CPU, the GPU has more computing units, but the units responsible for logic control only account for a small portion of it, so the GPU is more adept at dealing with simple tasks which require a large number of calculations and less adept at executing tasks which require a large number of logical judgments. The condition judgment affects the computation speed of the algorithm in the GPU, and the code addition table occupies a considerable part of the GPU video memory and bandwidth. Therefore, the above DGGS encoding is not efficient enough in topology query and hierarchical query on the GPU.

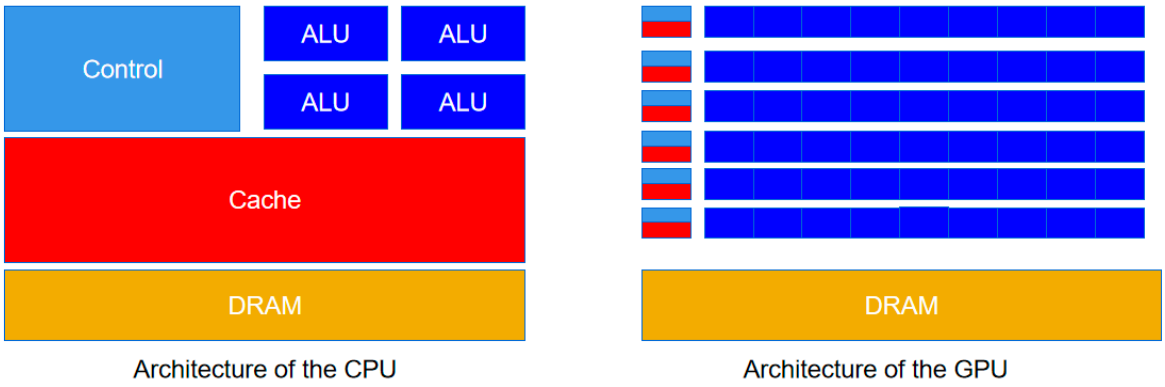

Figure 1. Differences in architecture between the CPU and the GPU. (The light blue rectangles represent the Control Unit, the dark blue rectangles represent the Arithmetic Logic Unit (AU), the red rectangles represent the Cache Memory, and the brown-yellow rectangles represent the Dynamic Random-Access Memory (DRAM)).

2.2. Integration Methods from Raster Data to DGGSs

In recent years, many scholars have also studied the integration algorithms of hexagonal grids and rasters. For example, Rui Wang [4] implemented a raster integration algorithm design based on a generalized hierarchical indexing method for the icosahedral mixed-aperture hexagonal grid, which can reduce the data volume by 38.5% compared to the pure-aperture algorithm design. Mingke Li [5], on the other hand, focused on the study of multi-resolution topographic map integration based on a hexagonal grid, which was achieved through the calculation of descriptive statistics, topographic parameters, and topographic indices for raster integration. Andrew Rawson [6] performed raster integration by resampling the raster surface using grid centroids and experimentally demonstrated the applicability of DGGSs as spatial data structures for maritime risk analysis. Jinxin Wang [19] proposed an algorithm for raster computation in the context of the Earth system space based on the spatial topological characteristics of 3D DGGS and the concept of dimensionality reduction. A Mahdavi-Amiri [20] presented a general hierarchical indexing mechanism for hexagonal cells resulting from the refinement of a triangular spherical polyhedral and applied it to the mapping of geospatial data to a DGGS. However, their method only considers the hierarchy of the hexagons themselves and does not encode the structure of rhombic tiles loaded with hexagonal grid cells. As a result, when the volume of data is very large, reading and searching the data become problems. Furthermore, this method does not limit the number of tiles by setting a threshold or tile merging, which generates a large amount of tile data when the amount of raster data is large. It requires the retrieval and reading of a large amount of tile data when performing the conversion of raster data to DGGSs, which has an impact on the integration efficiency.

As mentioned above, although many scholars have carried out research on DGGS-based raster data integration algorithms, their methods are designed for CPUs and are difficult to be implemented in GPUs, so they are limited by CPU performance and, usually, can only be used to convert low-resolution or localized high-resolution raster data to DGGSs.

2.3. GPU-Based Raster Data Integration and Organization

One of the main challenges that all digital Earth systems face is the sheer immensity of the amount of data available [11]. Massive raster data computation is difficult to load into a CPU, so many scholars choose to use GPU parallel computing to solve this problem. For example, Liheng Tan [21] proposed a method for the contour generation of DEM data based on a programmable GPU pipeline for a DGGS. Retief Lubbe [22] conducted an analysis of a GPU-based DEM parallel space partitioning algorithm, which provided guidance for further research in the field of GPU-based DEM collision detection and

its application in geotechnical engineering. Karnewar [23] implemented the processing and analysis of land remote sensing images based on the GPU. Lu Min [24] designed a CPU/GPU heterogeneous hybrid parallel model and performed the fast computation of raster data terrain factors based on this model. Lin Bo [25] implemented the GPU-based parallel computation of a linear integral convolution algorithm for ocean data visualization. Stojanovic [26] conducted the view analysis of digital elevation data based on the CUDA programming framework, etc.

Some scholars have also carried out research on the application of DGGSs to raster data organization, integration, and visualization in GPUs. For instance, M. J. Sherlock [27] proposed the method of mapping from hexagons to rhombuses and used a GPU for the efficient visualization of DGGS datasets. However, the essence of this research was to convert hexagonal DGGSs to rhombic DGGSs. However, in this method, when the amount of data was large, the volume of data being transmitted became a problem. In addition to this, the above-mentioned study used a coding system from the literature [18] to code hexagonal cells, so the hexagons also needed to be split during coding, and the spatial information of the extra vertices besides the centroid of the cell needed to be recorded. Xiaochuang Yao [28] designed a raster data integration method for hexagonal grids called HexTile on the Spark platform, which uses hexagonal tiles for raster data integration and adopts a distributed database for storage to solve the problems caused by map projection deformation. However, their method organizes the data by tiling each layer and does not use a rhombic tile structure, so the complex computation of neighborhood and hierarchical relationships by the hexagon itself inevitably affects the scheduling efficiency of the data in the integration process.

2.4. Summary of the Current Status of the Research

To sum up, the application of GPU technology to traditional raster data is quite mature, and the biggest difference between a raster and a hexagonal DGGS in terms of structure is that raster data are regular and can be passed to the GPU in the form of texture. Therefore, in this paper, we introduce a rhombic tile structure, so as to apply the research results of regular grids to a hexagonal DGGS, aiming to further improve the efficiency of the integration algorithm through the encoding and decoding mechanism of the DGGS and the design of GPU thread optimization.

3. Basic Idea and Overall Design

As stated above, the core idea of the method in this paper is to construct a hexagonal DGGS data structure based on rhombic tiles by combining the characteristics of hexagonal DGGSs with an icosahedron, so as to transform hexagonal DGGS data into a texture type acceptable for the GPU. On this basis, we designed the organization scheme of hexagonal DGGS data and the scheduling strategy of the corresponding raster data and created a conversion method from GPU-based raster data to hexagonal DGGS data through thread scheduling between the GPU and the CPU. The overall operation, which is divided into four steps, is shown in Figure 2.

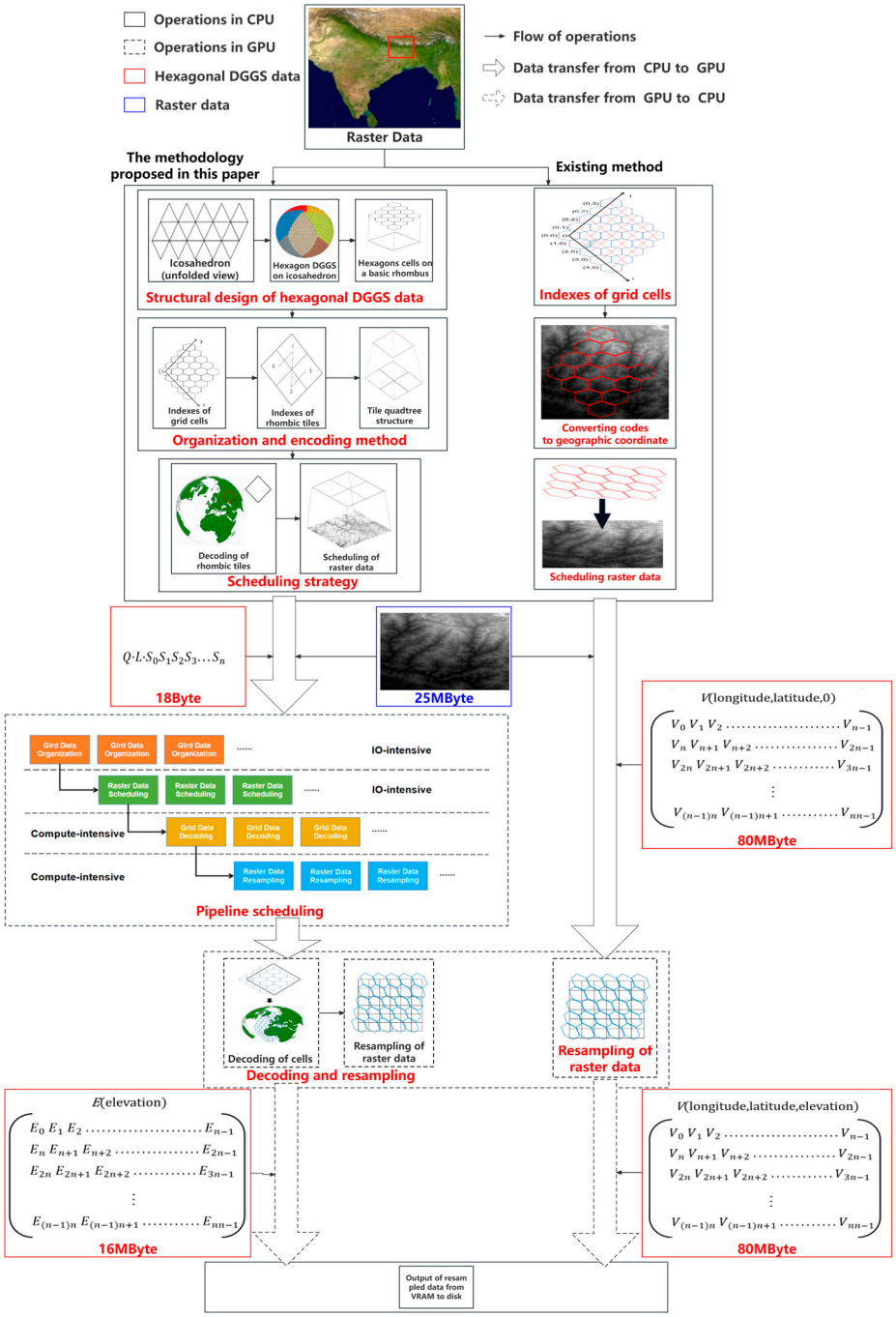

Figure 2. Pipeline scheduling.

(1) Structural Design: Given the distribution characteristics of a hexagonal DGGS on an icosahedron, the basic rhombic surfaces of the icosahedron are used to define a kind of data structure to organize hexagonal cells so as to achieve the organization of hexagons by rhombuses.

(2) Organization and Encoding Method: The rhombus-based hexagonal DGGS data organization scheme is then defined based on the constructed data structure, and the rhombus data as well as the hexagonal cells inside the rhombus are encoded as shown in Figure 2, where the hexagonal DGGS data are organized in a way that approximates the rhombic tiles of the rhombus, and the codes of the hexagonal cells are encoded in a relative coordinate system with respect to the tile codes of the rhombus.

(3) Scheduling Strategy and Resampling Algorithm: The geospatial extent of the rhombic tiles based on the level of hexagonal DGGS data and the encoding of the rhombic tiles is determined, and then the raster data are scheduled in the corresponding spatial extent based on the spatial extent of the rhombic tiles. After completing the scheduling, the rhombic tile data and the raster data corresponding to the spatial range are passed into the GPU, where the decoding of tile data is performed, and the raster data are resampled into the tile data.

(4) Pipeline Scheduling: Finally, in order to optimize the performance of the CPU and GPU, a parallel computing architecture of the GPU is designed to reduce its delay of the GPU in data reading and computation, and the operations in the whole process are divided into IO-intensive and compute-intensive parts according to the type of application, with IO-intensive applications executed in the CPU and compute-intensive applications executed in the GPU, so as to give full play to the parallel computing performance of the GPU, which is specifically used to achieve an efficient GPU-based conversion algorithm from raster data to hexagonal DGGS data.

4. Methodology

As the irregularity of hexagonal DGGSs is the fundamental reason why they are difficult to adapt to the GPU structure, the first problem to be solved in this research is how to express a hexagonal DGGS in the GPU. After this problem has been solved, we will proceed to design the scheduling strategy and resampling method for the raster data in the GPU, and, finally, through the asynchronous co-design between the GPU and the CPU, we will further enhance the computational performance of the GPU and create an efficient algorithm for the conversion of raster data to hexagonal DGGS data.

4.1. Representation of a Hexagonal DGGS in the GPU

4.1.1. The Hexagonal Organization Method Based on Rhombic Tiles

The grid dissection method based on polyhedral dissection is one of the basic dissection methods of DGGSs, where the triangular faces of an octahedron or icosahedron can be recursively subdivided into a hexagonal grid system [1]. In this paper, the ISEA4H generated by DGGRID, which is an open-source command-line application written in C++ [29], has been used, where A4 stands for an aperture size of 4. The aperture size is the ratio of the area occupied by coarse hexagonal cells to the area occupied by fine hexagonal cells [11]. In order to avoid the problem of the distribution of hexagonal cells on triangular surfaces, this study combines twenty triangular surfaces into ten rhombic surfaces in two groups, creating basic rhombic surfaces. There are $2^{level} \times 2^{level}$ uniformly distributed hexagonal cells on each basic rhombic surface, and in order for the hexagonal DGGS to cover the globe, there must exist 12 pentagons at the vertices of the ten rhombic faces. Taking the hexagonal grid with level 2 as an example, the icosahedral schematic and the arrangement of the hexagonal cells on the basic rhombic surface are shown in Figure 3.

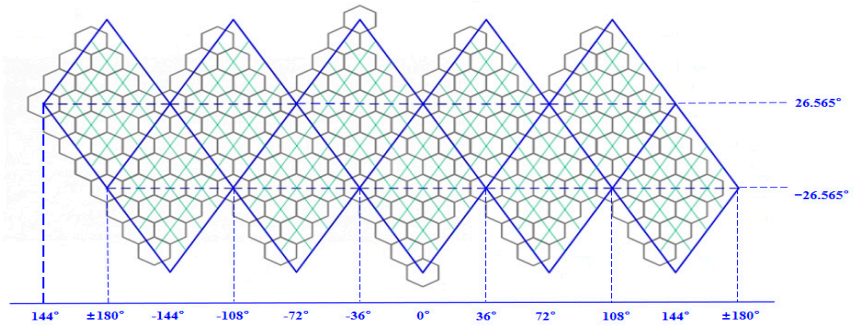

Figure 3. Arrangement of hexagonal cells.

If the centroids of the hexagonal cells are used to identify the hexagonal cells, the hexagonal grid can be transformed into a regular 2D rectangular array on the basic rhombic surface, which can be regarded as a kind of rhombic tile structure, as shown in Figure 4.

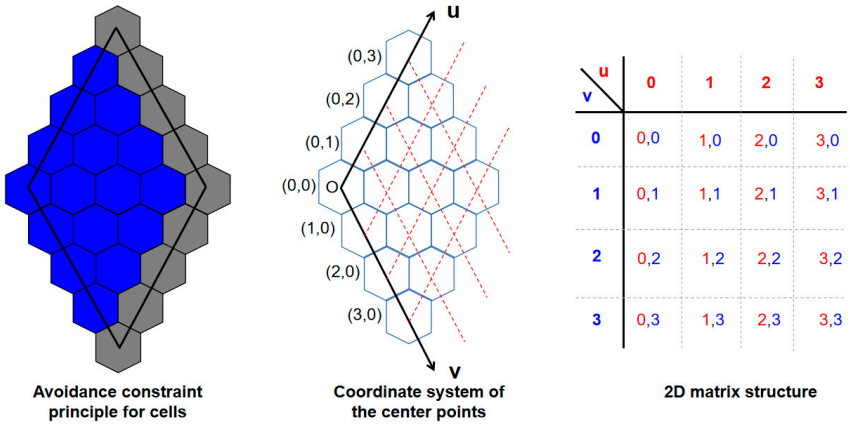

Figure 4. Organization of hexagonal cells based on rhombic tiles. (The blue hexagonal cells are the cells attributed to this basic rhombic surface, and the gray hexagonal cells are the cells that do not belong to the current basic rhombic surface).

The core idea of the method in this paper is to consider the underlying rhombic surface of a polyhedron as a kind of rhombic tile, construct its tree structure, and design the corresponding encoding and decoding mechanisms for the rhombic tiles, so as to organize the hexagonal cells into an approximately regular mesh structure, so that they could be received by the GPU in the form of a texture, based on which fast data access can be achieved. The hexagonal cells generated by the DGGRID are encoded to uniquely identify their position on the rhombic tiles, and the level of the tiles is the same as that of the hexagonal DGGS. The traditional tile quadtree structure is cut according to the level, but when the amount of data is large, this method affects the efficiency of the algorithm. Therefore, we designed a method for segmenting data according to their amount, and we set a threshold for it. When the amount of data reaches a certain threshold, the base rhombus is divided into four parts according to the threshold in accordance with the principle of quadtree slicing. From the above, it can be seen that, for the DGGS, the number of hexagonal cells is directly related to the level, so the threshold value can be formulated through the level of the DGGS. If one were to let the threshold level be M, then, when the grid level is less than M, there is a one-to-one linear relationship between the parent and child tiles, and only the hexagonal cells inside the tiles increase. When the grid level is greater than M, there is a one-to-four quadtree relationship between the parent and child tiles.

4.1.2. Encoding of Hexagonal Cells and Rhombic Tiles

There are two main ways to encode hexagonal cells on the DGGS: they are either given a unique ID or referred to by their basic rhombic surface number of the icosahedron (a rhombus made up of two triangular faces), numbered 0–9, and a two-dimensional coordinate on that quadrant, $q2di$ [30]. The $q2di$ is essentially a coordinate system code, which can be expressed in the form of a ternary, such as Cell(Q, R, C), where Q is the basic rhombic surface number of the icosahedron, R is the row number of the hexagonal cells on the underlying rhombic surface, and C is the column number of the cells on the underlying rhombic surface.

The method of unique ID encoding can be regarded as a simplification of $q2di$, while $q2di$ encoding of the hexagonal cells (Q, R, C) is implicit in the ID, and the encoding conversion method is shown in Equation (1).

$$ID = Q \times 2^{level} \times 2^{level} + R \times 2^{level} + C \quad (1)$$

Compared with the way of using unique IDs to encode the grid, *q*2*di* indexing is a choice related to the texture structure. However, the code of *q*2*di* is extremely lengthy when the grid level is high: assuming that *q*2*di* is to be used to encode the cells in the 18th level of the DGGS (with an approximate resolution of 30 m), the encoding of each cell will be up to 14 bits long, which will affect the efficiency of the encoding and decoding process and also make it difficult to adapt to the quadtree structure of the rhombic tiles. Therefore, we decided to shorten the number of bits encoded in the DGGS cells based on *q*2*di* by first encoding the rhombic tiles and later building the encoding coordinate system of the DGGS cells inside the rhombic tiles, in order to improve the efficiency of encoding and decoding.

Firstly, we encoded the rhombic tiles. From Section 4.1.1 of this paper, we know that the root nodes of the rhombic tiles are the ten basic rhombic surfaces of the icosahedron, which is a linear structure before the amount of data reaches the threshold, and a quadtree structure takes its place after the amount of data reaches the threshold. According to this characteristic, we defined a rhombic tile indexing method based on space-filling curves. The first byte of the code was Q, meaning the decimal form of the basic rhombus surface number, coded in the range 0–9, with the north and south poles denoted using N and S, respectively. The second byte was L, meaning the level of the grid, and, in order to avoid too many bits occupied by the level encoding, we used the thirty-sixth decimal form, ranged 0–36. The subsequent bytes were S, meaning the encoding based on the space-filling curve, which served to uniquely identify the subtiles obtained from the quadtree cut, as shown in Table 1.

Table 1. Encoding structure of rhombic tile.

First Character of the Code	Second Character of the Code	Subsequent Bytes of the Code
Decimal form of the basic rhombic surface numbering: 0–9, N, S.	Level of the grid, using thirty-sixth decimal representation.	Unique identification of the binary code using space-filling curves.

If one were to let the code of the rhombic tile be the Morton curve, then the code of the rhombic tile quadtree would be that shown in Figure 5.

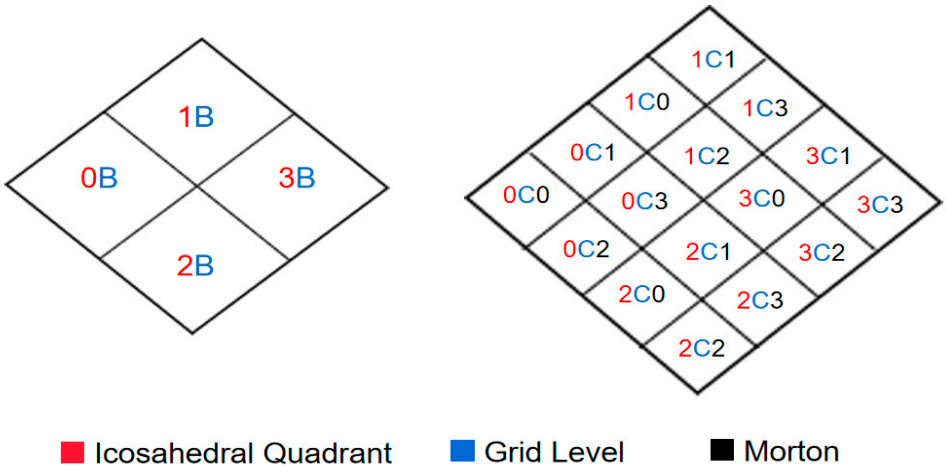

Figure 5. Rhombic tile encoding structure.

According to the relationship between the hexagonal DGGS and the rhombic tiles, the *q*2*di* indexing range of the hexagonal cells in each rhombic tile can be calculated based on the tile code. If one were to let the grid level be $M(M \in N^+)$, then the index of the rhombic tile would be $Index_r = Q_r \cdot L \cdot S$, where Q is the basic rhombic surface number, L is the grid level, S is the space-filling curve code, and the subscript r represents the rhombic tile. For

the hexagonal cells inside the tile, $Q_{cell} = Q_r$. If the cut threshold of the quadtree is not reached, the encoding range of the hexagonal cells is $R_{range} = C_{range} = [0, 2^L), L \leq M$, where the subscript *range* represents the encoding range, R represents the row where the hexagonal cell resides, and C represents the column where the hexagonal cell resides. When the cut threshold of the quadtree is reached, R_{range} and C_{range} can be calculated using Equations (2) and (3).

$$R_{range} = \begin{cases} [0, 2^{L-1}), S = 0,\ 1 \\ [2^{L-1}, 2^L), S = 2,\ 3 \end{cases}, L = M + 1 \tag{2}$$

$$C_{range} = \begin{cases} [0, 2^{L-1}), S = 0,\ 2 \\ [2^{L-1}, 2^L), S = 1,\ 3 \end{cases}, L = M + 1 \tag{3}$$

In order to avoid the redundancy of the *q*2*di* code, a relative coordinate system can be defined so that the code of the hexagonal cells is transferred from the global *q*2*di* code to the *q*2*di* encoding relative to the rhombic tiles. The process can be divided into the following five steps, as shown in Figure 6, the pseudo-code of which is given in Figure A1.

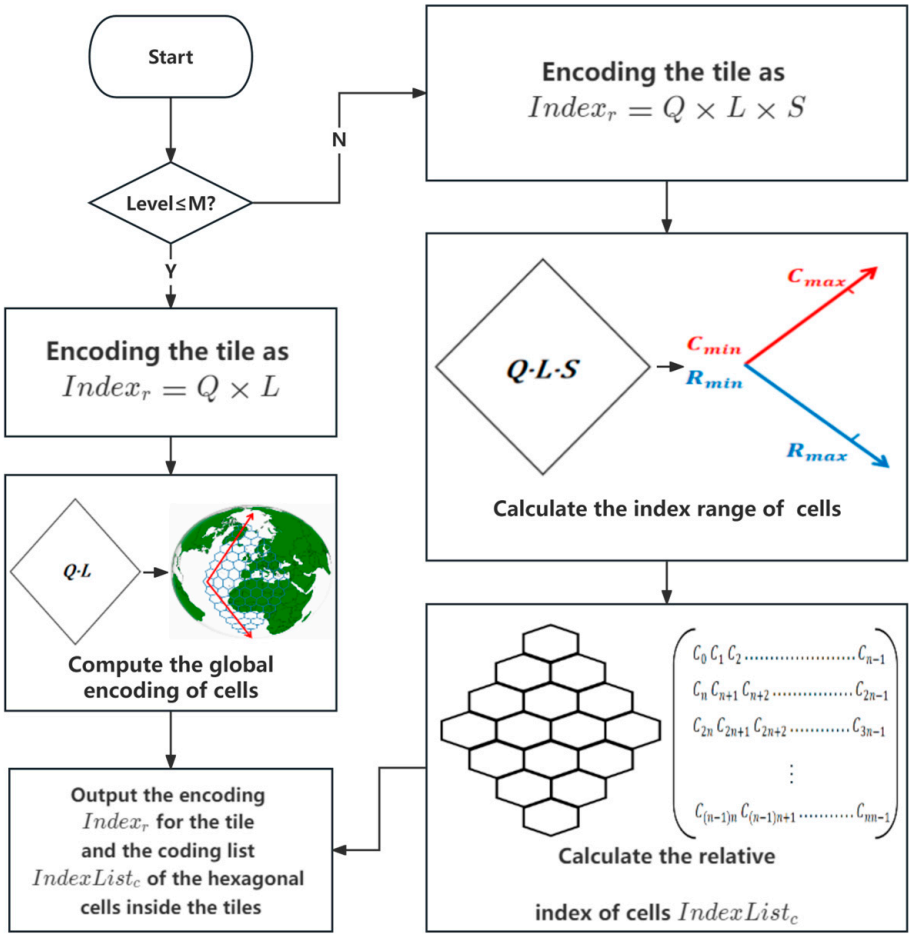

Figure 6. Encoding process for hexagonal cells.

- Step1: Determine whether the grid level L is greater than the threshold M.
- Step2: If the grid level L is greater than M, skip to step 4, and, if the grid level L is less than or equal to M, the rhombic tiles are encoded without the space-filling curve code S. They will simply be encoded as $Index_r = Q_r \cdot L$.
- Step3: The hexagonal cells inside the rhombic tiles are global coordinates when the grid level is less than the threshold M, and $Q_c = Q_r$.

- Step4: If the grid level is greater than M, based on the tile code $Index_r = Q_r \cdot L \cdot S$, the encoding range R_{range} (R_{min}, R_{max}) and C_{range} (C_{min}, C_{max}) of the hexagonal cells inside the rhombic tile is first calculated using Equations (2) and (3). Then, the encoding of the hexagonal cells with respect to the coordinate values of the rhombic tile origin is calculated: $Q_c = Q_r$, $R_c = R_c - R_{min}$, *and* $C_c = C_c - C_{min}$.
- Step5: Repeat the above steps until all the encodings of the hexagonal cells have been converted to the encodings relative to the rhombic tile. Output the list of converted encodings, i.e., $IndexList_c = \{(Q_{c_1}, R_{c_1}, C_{c_1}), \{(Q_{c_2}, R_{c_2}, C_{c_2}), \{(Q_{c_3}, R_{c_3}, C_{c_3}), \ldots, \{(Q_{c_n}, R_{c_n}, C_{c_n})\}$, $n \in N^+$.

4.1.3. Decoding of Hexagonal DGGS Data in the GPU

The conversion between the *q*2*di* codes and the geographic coordinates is based on Snyder equal-area map projection [31]. When converting the codes of hexagonal cells to geographic coordinates, the first step is to convert the codes of the hexagonal cells from those relative to the rhombic tiles to the global *q*2*di* codes. If one were to let the encoding of the hexagonal cells relative to the rhombic tiles be Cell(Q_r, R_r, C_r), then, in the *q*2*di* coordinate system, the basic rhombic surface of the hexagonal cell would be $Q_{q2di} = Q_r$, the row number would be $R_{q2di} = R_r + R_{min}$, the column number would be $C_{q2di} = C_r + C_{min}$. R_{min}, and C_{min} could be decoded from the rhombic tile encoding of where the hexagonal cell is located. After that, the *q*2*di* code is decoded and converted to geographic coordinates by Snyder equal-area map projection.

In this paper, the encodings of hexagonal cells are realized based on the encoding of rhombic tiles, so only the encodings of rhombic tiles need to be transmitted during data transmission, after which the encodings of all hexagonal cells can be calculated based on the encoding of rhombic tiles and then decoded using the Snyder equal-area map projection. Due to this feature, when the number of hexagonal cells is large, parallel computation using the GPU can realize a great improvement in decoding efficiency.

Therefore, in this paper, the decoding operation of the hexagonal cells is designed to be executed in the GPU, and only the encodings of rhombic tiles are passed into the GPU without considering the decoding of specific hexagonal cells, and the vertex shader is used in the GPU to perform parallel computation to code the hexagonal cells and convert them to geographic coordinates. The whole operation breaks into the following four steps:

- Step1: Generate a list of encodings in the CPU for the rhombic tiles whose spatial extent intersects the target spatial extent space, then traverse the list and pass the encodings to the GPU.
- Step2: After the GPU receives the codes, it first decodes the rhombic tile to obtain the number Q of the underlying rhombic surface where the rhombic tile is located and carries out some calculations to obtain the relative coordinate ranges R_{range} and C_{range} of the hexagonal cells inside the tile.
- Step3: Based on the calculated basic rhombic surface numbers and the relative coordinate ranges, the hexagonal cells inside the rhombic tiles are decoded to obtain the *q*2*di* codes of the hexagonal cells.
- Step4: The *q*2*di* codes are converted to geospatial coordinates, and the next loop is performed until all hexagonal cell codes have been converted to geospatial coordinates. Then, the next pieces of rhombic tile data are decoded, until all tile data have been decoded, as shown in Figure 7. A pseudo-code that describes the process of decoding is given in Figure A2.

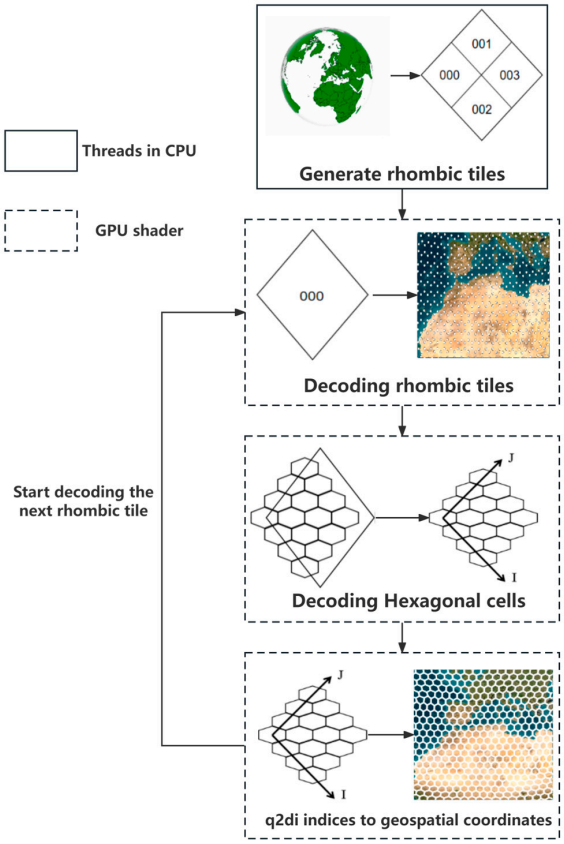

Figure 7. Decoding process for hexagonal cells.

4.2. Raster Data Scheduling and Resampling in the GPU

4.2.1. The Scheduling Strategy for Raster Data in the GPU

As previously mentioned, in this paper, DGG data are organized in rhombic tiles based on the ten basic rhombic surfaces in the positive icosahedra, and, as raster data are regularly arranged and the range of geographic coordinates between each block of raster data is consistent, this research adopts a rhombic tile-based approach to look up raster data for spatial data scheduling. Specifically, it is a method of traversing rhombic tiles based on the space-filling curve, then calculating the space range of each rhombic tile and scheduling the raster data that partially or fully lie within the space range of rhombic tiles, and, finally, assigning values to the hexagonal cells inside the tile data on this basis. This method reduces the number of projection calculations and eliminates the case of crossing tiles compared to raster-based rhombic tile data finding.

The scheduling strategy for raster data whose spatial extent intersects the spatial extent of the rhombic tile is shown in Figure 8. First, the spatial extent of the rhombic tile and the initial rhombic surface number where it is located are calculated based on the tile code. Afterwards, raster data with similar extents are dispatched based on the spatial extent of the rhombic tiles. When performing DGGS-based raster data integration, the resolution of the DGGS is usually similar to the raster resolution, but, since the ISEA4H used in this paper is an equal-accumulation grid and the metric unit of the rhombic tiles is in meters while the metric units of the raster data are latitude and longitude, the spatial relationship between the rhombic tiles and the raster data exists differently at different latitudes. Taking global Aster 30 m digital elevation data as an example, its corresponding DGGS level is 18. As shown in Figure 9, the relationship between rhombic tiles and raster data blocks can be categorized into the following six kinds, but the number of raster data blocks around each rhombic tile is not more than nine. Therefore, each rhombic tile is dispatched accordingly with a 3×3 raster combination block.

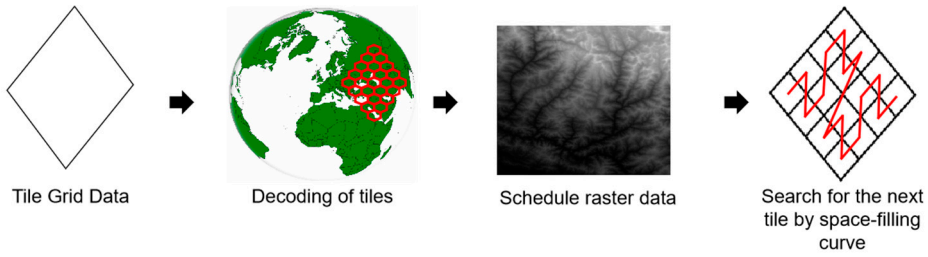

Figure 8. Raster data scheduling strategy.

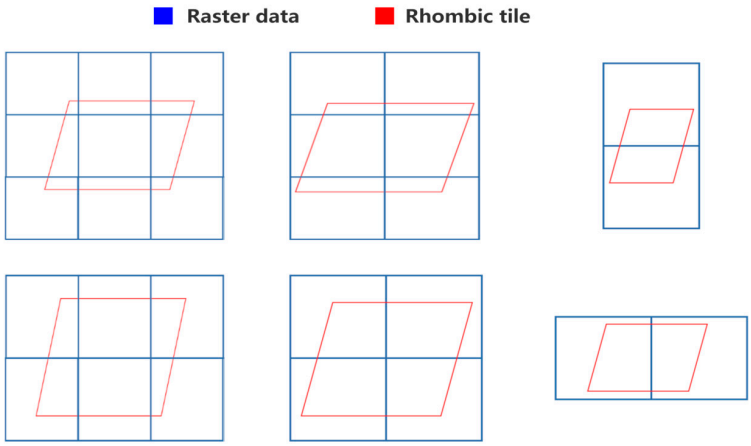

Figure 9. Spatial relationship between raster data and rhombic tile data.

After completing the assignment of grid data within a rhombic tile, the next tile to be assigned is determined based on the space-filling curve, with the current tile and the next tile usually being adjacent to one another. Therefore, the spatial extent of the next tile can be calculated based on the method above, after which the 3×3 grid combination block is updated accordingly by comparing the spatial relationship between the next tile data and the current tile data. The raster data that are still within the range of the grid data are kept, and the raster data that are beyond the range of the grid data are updated, to avoid reading raster data repeatedly, as shown in Figure 10.

0 (lon-1,lat+1)	1 (lon,lat+1)	2 (lon+1,lat+1)
3 (lon-1,lat)	4 (lon,lat)	5 (lon+1,lat)
6 (lon-1,lat-1)	7 (lon,lat-1)	8 (lon+1,lat-1)

(a)

Remove 0	Reserve 1--0′	Reserve 2--1′	Add 2′
3	4--3′	5--4′	5′
6	7--6′	8--7′	8′

(b)

Figure 10. Raster data scheduling strategy: (**a**) Schematic diagram of the first raster data scheduling; (**b**) Update strategy for raster data.

After completing a grid data assignment, the read raster data are cleared from the video memory, and the subsequent tile data are added to the video memory. The relationships between the subsequent tile data and the completed tile data can be divided into nine types, as shown in Figure 10a. For example, if the geographic coordinate range of the subsequent tile data is equal to that of the completed tile data, this corresponds to case 4 in

Figure 10a, where no raster combination block update is required. If the subsequent tile data are increased by 1° in longitude compared to the completed tile data and the latitude remains unchanged, it corresponds to case 5 in Figure 10a. At this time, the data of blocks 0, 3, and 6 of the raster combination blocks are rejected, and blocks 1, 4, 7, 2, 5, and 8 are shifted left by one position to blocks 0′, 3′, 6′, 1′, 4′, and 7′, and the data of blocks 2′, 5′, and 8′ will be reread, as shown in Figure 10b. A similar strategy can be used for the remaining cases. If the geographic coordinate range relationship between the subsequent tile data and the completed tile data is not among the nine cases in Figure 10a, the 3 × 3 raster combination blocks will be searched again based on the geographic coordinate range of the subsequent tile data.

4.2.2. Raster Data Resampling Based on the Hexagonal Grid

Considering the fact that raster data are mostly organized in the form of a positive quadrilateral, while hexagonal grid data are organized in the form of a positive hexagon, resampling will be required if integration between the two types of data is to be achieved after completing raster data scheduling. For raster data, currently, the common resampling methods include neighborhood interpolation, bilinear interpolation, and bicubic interpolation. Among them, the bilinear interpolation method is a resampling method in which four coordinate points in the data are known and linearly interpolated in two directions, respectively, horizontal and vertical, which maintains a higher accuracy than the neighborhood interpolation method and enjoys a higher computational efficiency than higher-order interpolation algorithms such as bicubic interpolation, with insignificant differences in accuracy [32]. For the above reason, this paper attempts to realize the integration from raster data to hexagonal grid data based on the bilinear interpolation method.

The principle of bilinear interpolation is shown in Figure 11. The red point in Figure 11b is the center point of the hexagonal cell inside the tile data, and the black point is the center point of the raster data cell. The specific steps are as follows: first, the geospatial extent of the rhombic tile is calculated based on the encoding of the rhombic tile, and raster data that are partially or fully located within the geospatial extent of the rhombic tile are scheduled. After the scheduling is completed, the bilinear interpolation value of the positive hexagonal grid is calculated based on the obtained raster data blocks. If one were to let the coordinates of the center point of the hexagonal grid be (x, y), the horizontal partition ratio be μ and $1 - \mu$, and the vertical partition ratio be λ and $1 - \lambda$, $F(x, y)$ would denote the original value at the center (x, y) of the positive quadrilateral grid, and $I(x, y)$ would denote the interpolated value at the point (x, y). Then, the following could be obtained from the formula of linear interpolation.

$$\begin{aligned} I(x_1, y) &= \lambda F(x_1, y_2) + (1 - \lambda) F(x_1, y_1) \\ I(x_2, y) &= \lambda F(x_2, y_2) + (1 - \lambda) F(x_2, y_1) \\ I(x, y) &= \mu I(x_2, y) + (1 - \mu) I(x_1, y) \end{aligned} \tag{4}$$

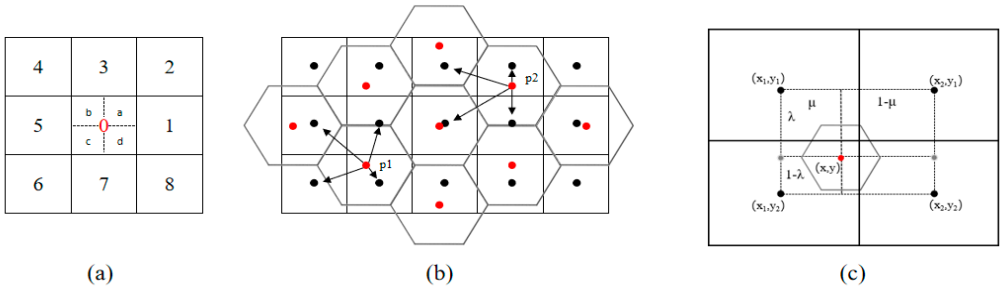

Figure 11. Principle of bilinear interpolation of raster data: (**a**) Principle of bilinear interpolation; (**b**) Finding the nearest raster point from the center of a hexagonal cell; (**c**) Interpolate the center of the hexagonal cell based on the coordinates of the four nearest raster center points.

From this, the bilinear interpolation formula for the center of the hexagonal grid can be derived, as in Equation (5).

$$I(x,y) = \mu\lambda F(x_2,y_2) + \mu(1-\lambda)F(x_2,y_1) + (1-\mu)\lambda F(x_1,y_2) + (1-\mu)(1-\lambda)F(x_1,y_1) \quad (5)$$

In this research, the raster data resampling algorithm is applied in the GPU in order to avoid the performance problem caused by transferring arrays larger than the maximum number of threads to the GPU in high-resolution raster integration. To be exact, this research adopts the method of mesh grid–stride loops (GSLs) to execute the resampling of raster data in the CUDA. During the data solving process, the grid–stride loop is implemented in the entire threaded grid by means of the jump traversal of the one-dimensional data array by one grid width at a time, that is, by creating a loop which takes the thread grid through the entire array, in which case, each thread must move to the next unprocessed location after one loop iteration. If one were to suppose that there were two grids in a thread, with four blocks in each grid, the grid–stride loops would be executed as shown in Figure 12.

Figure 12. Grid–stride loops. (Dark blue rectangles represent Grids in threads. light blue and white rectangles represent Blocks in a Grid. orange rectangles represent data blocks).

4.3. Asynchronous Collaboration Method of the CPU and the GPU

For the research goal of this paper, efficient CPU and GPU asynchronous collaboration is essential to further realize GPU-based algorithm acceleration. After the resampling scheme is completed, threads can be allocated according to the thread block into four steps: the decoding of tiles, the scheduling of raster data, the resampling of raster data, and the output of tiles. Threads are allocated in accordance with the tile data structure designed in this study, and each rhombic tile serves as the basis of one thread, which is executed in a concurrent manner. By analyzing the above process, it can be found that the four steps in the process have different time complexities, as shown in Table 2.

Table 2. Time complexity of different steps.

The Output of Tiles	The Scheduling of Raster Data	The Decoding of Tiles	The Resampling of Raster Data
O(n)	O(n log n)	O(n)	$O(n^2)$

Different time complexities will lead to conflicts between threads, resulting in serious problems such as data loss and duplicate writes, so inter-thread synchronization is needed. However, if the traditional thread lock mechanism is used in this process, it will lead to a long waiting time for unlocked threads, which will cause performance degradation of

the GPU. For example, while the resampling thread for the raster data is resampling raster data into a rhombic tile, if the scheduling thread for the raster data wants to transmit the next rhombic tile and the raster data which correspond to the spatial extent of the rhombic tile, the scheduling thread has to wait for the resampling thread to complete the operation, which causes unnecessary delays. Therefore, in this paper, based on the thread lock, we set up a buffer between threads and send signals to synchronize four threads by means of thread communication. The steps are shown in Figure 13.

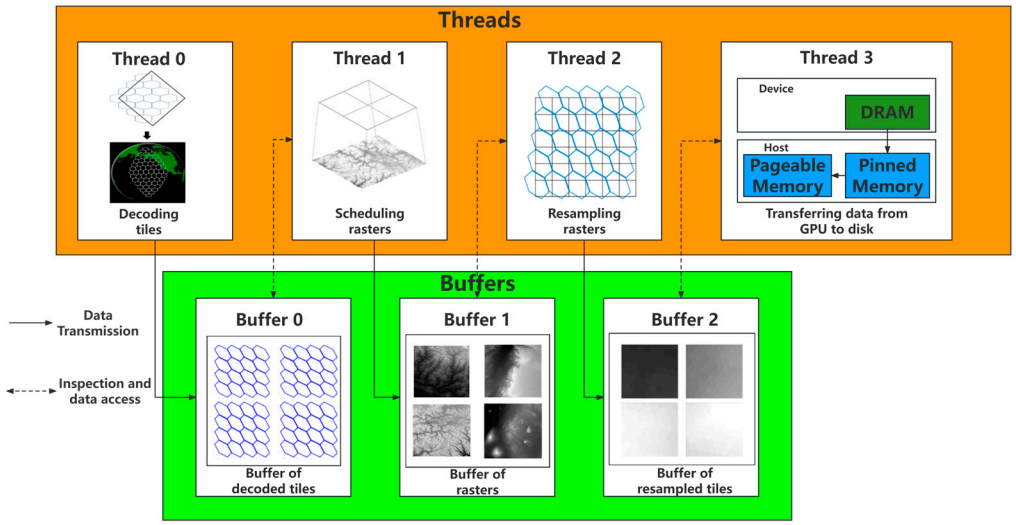

Figure 13. Thread buffer design.

- Step1: After hexagonal DGGS data are transferred from the CPU to the GPU in the form of a texture, they are first decoded to obtain their spatial extent as well as the geographic coordinates of the hexagonal cells. Whenever the decoding of tile data is completed, it is saved in the tile data-decoding buffer, and a signal is sent to activate the thread used for raster scheduling.
- Step2: The raster-scheduling thread first traverses the tile data-decoding buffer, and, whenever it obtains tile data, it schedules the raster data in the corresponding spatial range, saves the completed raster data and tile data in the raster-resampling buffer, and sends a signal to activate the raster-resampling thread at the beginning of the thread.
- Step3: The raster-resampling thread continuously checks the raster data buffer for unfinished raster data and tile data and then resamples the raster data and assigns values to the tile data after obtaining the data. After the assignment is completed, the raster-resampling thread saves the tile data to the data output buffer and sends a signal to activate the data output thread at the beginning of the thread.
- Step4: The data output thread continuously checks the raster-resampling buffer and sets the threshold. When the amount of data in the buffer reaches the threshold, all the tiles in the buffer will be output from the GPU to the disk to avoid the performance loss caused by multiple transfers.

When the tile data-decoding thread completes the decoding of all the tile data, it will automatically stop and send a signal to the grid-scheduling thread. The grid-scheduling thread that receives the signal will stop when the tile data-decoding buffer is empty and send a signal to the grid-resampling thread, which, upon receipt of the signal, will stop after completing the resampling of all the data in the buffer, and it will send a signal to the data output thread. When the raster-resampling buffer is empty, it means that the raster integration algorithm has been fully executed, and all the data have been transferred from the GPU to the disk.

As it can be seen from the above, this operation no longer passes tile data and raster data as parameters between threads but saves them in the thread buffer to achieve syn-

chronization between the threads in the way of thread communication. Each thread in this process is activated by the previous one, and it only needs to check whether there are any unprocessed data in the buffer after activation, without directly exchanging information between threads, thus avoiding the performance impact caused by using the traditional thread-locking method, and, eventually, an efficient buffer-based inter-thread synchronization strategy is successfully implemented.

5. Results

In order to verify the accuracy and efficiency of the proposed algorithm, raster data integration experiments based on an 18-level global hexagonal discrete grid were designed and conducted in this research, using Aster 30 m digital elevation data on a global scale. The area of each hexagonal cell was 742.25 m^2, and the distance between the centroids of the hexagonal cells was 33.8 m under the 18-level grid. The experimental environment was a desktop computer with Microsoft Windows 10 Professional installed, equipped with the Intel(R) Core (TM) i7-10875H CPU at 2.30 GHz, NVIDIA GeForce RTX 2060 graphics card, 32 GB of RAM, and 1 tb hard drive at 7200 rpm. The program was compiled using Visual c++ 2019.

The experimental results were evaluated in terms of both integration accuracy and integration efficiency, in comparison with the hexagonal connectivity map [20], i.e., a CPU-based raster data integration algorithm. In order to determine the integration accuracy, we selected several regions with fragmented terrain and traversed each data cell in the raster data in the region to search for the DGG cells in the corresponding position according to their latitude and longitude, then compared their attribute values, and, if the values were the same, they were considered to meet the accuracy criteria. This process continued until all traversals had been completed. Since the integration efficiency of raster data is influenced by several factors, such as the threshold for tile splitting, the combination of threads in the CUDA, the decoding method of tile encodings, etc., we planned to conduct the experiments starting from the influencing factors of raster data integration efficiency and, after obtaining the most efficient solution, compare the efficiency with that of the CPU-based raster data integration algorithm.

5.1. Integration Accuracy of the Algorithm

To prove the accuracy of raster data integration, three areas with a relatively complex topography, namely, the Yarlung Tsangpo Grand Canyon, the Dead Sea, and Mount Everest, were selected in this paper to validate the resampled gridded raster data within the area. As stated in Section 4.2.2 of this paper, the interpolation method used in the resampling of raster data is bilinear interpolation. Therefore, in order to ensure the rationality of the verification results, bilinear interpolation was also used to resample the DGGS data to the same resolution as the raster data, and then the elevation value of each of the same raster points in the DGGS data and the raster data was compared, and the average error was calculated as well. Table 3 lists the calculation results.

From the results of the integration accuracy verification experiments, it can be seen that the proposed algorithm in this paper can effectively guarantee the accuracy of data integration when raster data integration is performed. Taking the Yarlung Tsangpo Grand Canyon region as an example, the resampled Aster 30 m raster data had a total of 104,857,600 hexagonal cells, and the average error between the DGGS data integrated with the raster data and the original raster data in terms of the elevation value was 1.068 m. The different integration accuracies between the raster data in the three different regions were due to the bilinear interpolation method, which is essentially a low-pass filter. The Dead Sea has little topographic relief, so the raster was dominated by low-frequency signals and, therefore, had a higher integration accuracy. The Great Canyon of Yarlung Tsangpo and Mount Everest have more topographic relief and more high-frequency signals in the rasters, so the integration accuracy in their case was lower compared to the Dead Sea.

Table 3. Integration accuracy verification results.

Location	Space Range	Number of Hexagonal Cells	Average Error of Grid Integration (m)	Max Error of Grid Integration (m)	Min Error of Grid Integration (m)
Great Canyon of Yarlung Tsangpo	30–35°N, 90–95°E	104,857,600	1.068	120.530	0.412
Dead Sea	30–32°N, 36–36°E	8,388,608	0.859	0.747	0.316
Mount Everest	27–28°N, 86–87°E	4,194,304	1.481	209.548	0.383
Whole Earth	0°N–90°S, 0°E–180°W	687,194,767,362	0.257	209.548	0.001

5.2. Integration Efficiency of the Algorithm

5.2.1. Scheduling Schemes for the Raster Data and the Tile Data

In our experiment, firstly, the threshold of rhombic tiles' segmentation was tested. The threshold represents the maximum number of hexagonal cells that can be stored inside a tile. By testing the indexing speed of tile encoding and the IO speed of the raster data, the threshold of tile segmentation that was most conducive to efficient data transmission was calculated. Since rhombic tiles store hexagonal cells internally, the amount of data must follow the distribution law of hexagonal cells, which means that the amount of data must be $2^{level} \times 2^{level}$. The data-scheduling steps in the experiment are specified in parahraph 4, Section 1, of this paper, and the experimental results are shown in Figure 14.

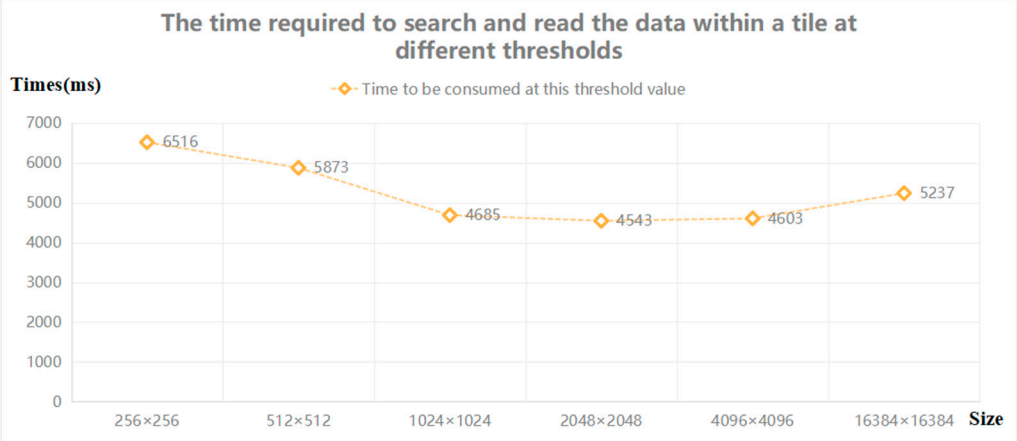

Figure 14. Experiment to determine the threshold of tile splitting.

It can be seen from the experimental results that the segmentation threshold of tile data is closely related to the search and scheduling efficiency of the raster data. If the threshold is too small, it will greatly increase the number of operations for retrieval and scheduling, which will affect the efficiency of data indexing. On the other hand, if the threshold is too large, it will increase the amount of transmission per raster data block, causing an increase in the time for raster data to be read. The experiments show that the IO efficiency of the raster data reaches the highest value when 2048 × 2048 hexagonal cells are decoded in each rhombic tile, at which time the grid level is 11. Therefore, the threshold M in Section 4.1 of this paper is 11.

Though the scheduling strategy between individual tile data and raster data has been determined in Section 4.2.1 of this paper, but, for the full tile data, traversal through space-filling curves is still required. Therefore, the impact of different common space-filling curves on the efficiency of the algorithm has to be evaluated. The raster data used for the evaluation are the global Aster 30 m digital elevation data, which correspond to tiles with a level of 18. The evaluation method is to traverse the same tile data point based on

different space-filling curves and record the raster data that are scheduled in each tile data. After completing the traversal, the average number of times that each raster data point is scheduled in the traversal process is counted. The experimental results are shown in Figure 15.

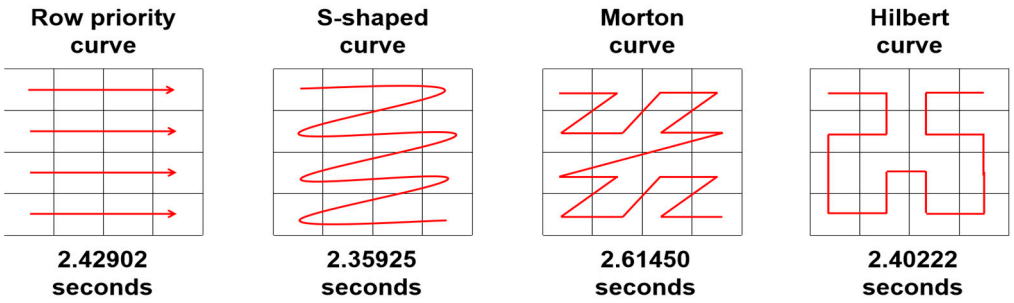

Figure 15. Comparison of efficiency of space-filling curves (times).

From the experimental results, it can be seen that the S-shaped space-filling curve has a higher searching efficiency with fewer reads per raster data on average compared to other curves when traversing tiles; therefore, the S-shaped space-filling curve is chosen as the basis for tile traversal, which improves the efficiency of raster data scheduling.

5.2.2. Thread Combination in the GPU

When CUDA is executed, the Kernel function in the Host program is executed on the GPU according to the concept of grid (thread grid). A Kernel function corresponds to a grid. When a task is to be executed, the grid assigns the task to part of a block (thread block), which contains multiple threads, and the block assigns the task to different threads in it and performs concurrent computations. In order to achieve the highest degree of concurrency, it is necessary to conduct experiments to find out the best sizes of block and grid that are conducive to the highest efficiency. Therefore, in this paper, "gridsize" denotes the size of the grid, and "blocksize" denotes the size of the block in CUDA. We conducted a comparison experiment on the computation efficiency for different combinations of gridsize $\times$ blocksize, and the experimental results are shown in Figure 16.

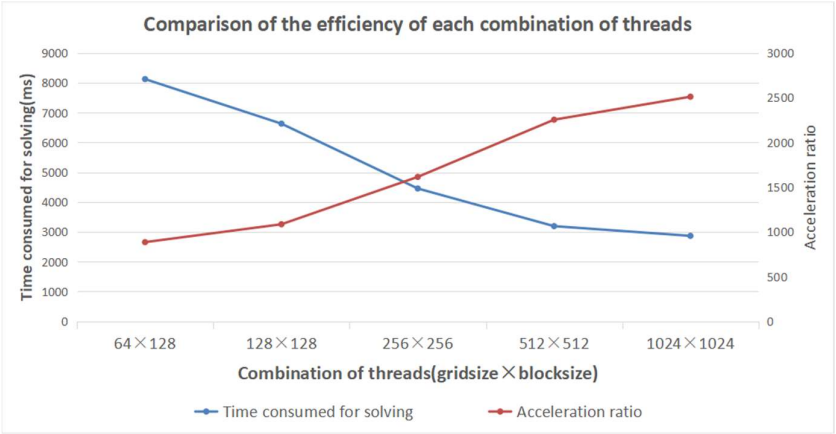

Figure 16. Efficiency comparison for each thread combination block.

From the experimental results, it can be seen that, when the thread combination reaches 256 $\times$ 256, the solution speedup ratio reaches the highest peak, and the subsequent increase in threads can no longer improve the solution efficiency, so a gridsize $\times$ blocksize of 256 $\times$ 256 is chosen as the thread combination scheme in this paper.

5.2.3. Decoding Efficiency Comparison between the CPU and the GPU

As described in Section 4.1.3 of this paper, theoretically performing the decoding of the hexagonal DGGS data in the GPU has a higher efficiency compared to performing it in the CPU, but the actual results still need to be verified by experiments. The comparison of decoding efficiency consists of two parts: the first part is the efficiency of the conversion from encoding to geographic coordinates, and the second part is the efficiency of the data transfer from the CPU to the GPU. As the transmission efficiency of the hexagonal DGGS data has already been verified in Section 5.2.1 of this paper, an experiment is conducted for validating the decoding efficiency only.

The raster data used for the experiment are the global Aster 30 m digital elevation data, and the experimental method is to first encode the hexagonal DGGS data based on rhombic tile organization on the 15–18 level, respectively, and then decode and record the time of operation in the CPU and the GPU, respectively, where the CPU-based algorithm uses the hexagonal connectivity map [20], and the GPU-based algorithm uses the algorithm proposed in this paper. The experimental results are shown in Figure 17.

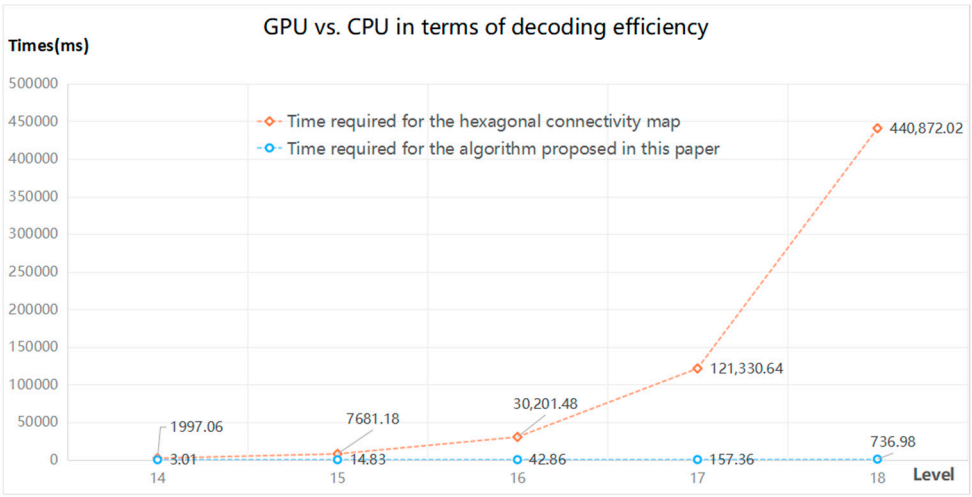

Figure 17. CPU decoding efficiency vs. GPU decoding efficiency.

The experimental results show that the efficiency of decoding hexagonal DGGS data in the GPU is much higher than that in the CPU. The fundamental reason is that the decoding of hexagonal DGGS data is a computation-intensive application, and GPUs, with more processing units and a higher memory bandwidth than traditional central processing units (CPUs), are able to increase the efficiency of computationally intensive applications through multi-threaded parallel computing.

5.2.4. The Overall Efficiency

Based on the above experiments, the optimal combination scheme of the algorithm in an NVIDIA GeForce RTX 2060 graphics card can be derived: that is, each rhombic tile can be decoded in the GPU to obtain 2048 × 2048 hexagonal cells, on the basis of which the scheduling of grid data is achieved using the S-shaped space-filling curve, and the combination of the grid and the block is defined as 256 × 256 in CUDA. The time consumed by each part of the process of GPU-based transmission from raster data to the hexagonal DGGS using the above parameters is shown in Figure 18, and the *x*-axis is the level of tiles.

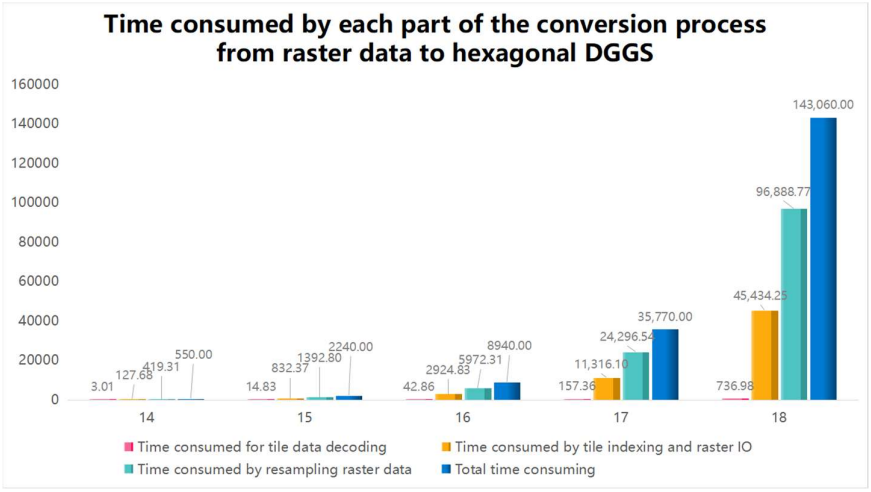

Figure 18. Time consumed by each part of the algorithmic flow.

Based on this scheme, a comparison of the overall efficiency between the proposed algorithm and the traditional CPU-based raster data integration algorithm is conducted. The CPU-based integration algorithm uses hexagonal connectivity maps [20], and the comparison results are shown in Figure 19.

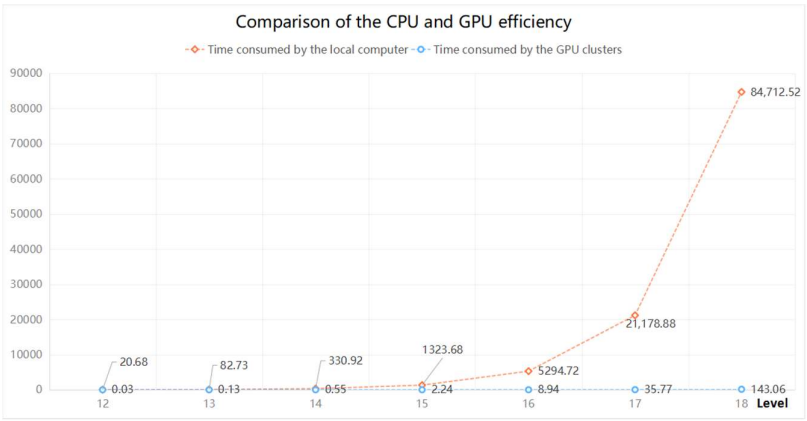

Figure 19. Overall efficiency comparison.

According to the overall experimental results, the proposed algorithm achieved significant efficiency improvements while ensuring the accuracy of data integration, as compared with the traditional CPU-based raster data integration algorithm. Take the 18th-level grid as an example, the complete integration of the global raster data using the CPU takes approximately 23.53 h, while the complete integration of the global raster data using the GPU takes approximately 143.0624 s, which is about 600 times more efficient in computation.

6. Discussion

The discussion section of this paper is divided into three parts: first, the comparison of the encoding and decoding methods proposed in this paper with the existing hexagonal DGGS encoding methods; second, the comparison between the algorithm proposed in this paper and the existing methods in terms of GPU application; and third, the scalability of the method proposed in this paper in the field of DGGSs.

6.1. Encoding and Decoding Methods

DGGS is an encoding-centered digital Earth framework, and the efficiency of encoding and decoding have a great impact on all DGGS-based operations. The encoding

and decoding mechanisms are also part of the main difference between the integration algorithm proposed in this paper and other existing algorithms. Three commonly used hexagonal DGGS coding and decoding methods are selected here for comparison, which are PYXIS [13], HQBS [12], and hierarchical grid conversion [18].

PYXIS is a coding method designed for A3H DGGS, and it does not generalize to other types of DGGS. It divides the sphere into 32 regions, each modeled by a multi-resolution sequence of finite planar hexagonal grids. Operations based on PYXIS encoding need to be performed through pre-computed addition tables, and, due to the inconsistency between different levels, PYXIS needs to make a lot of logical judgments when performing neighborhood query, hierarchical query, parent–child relationship query, and other operations on cells.

HQBS is a method of encoding hexagonal cells using consecutive quadratic numbers, each bit of which is encoded in the range {0, 1, 2, 3}, and it sets an origin for the hexagonal cells, which is noted as 0. The rest of the encoding for each cell records the direction and distance between it and the origin. When performing operations based on HQBS encoding, it is necessary to set up the addition and multiplication tables in a pre-computed form and perform the encoded calculations by looking up the tables to obtain the desired results.

The core idea of hierarchical grid conversion is to insert new vertices or edges or removing the existing ones to realize the conversion between different graphics. Therefore, the coding method in this paper serves this purpose, which means that additional vertex information must be recorded in the encoding in order to ensure that the converted graph can be converted back to the original graph. Furthermore, if this coding method is applied to the digital Earth, for instance, to construct a mapping from hexagonal DGGS to rhombic DGGS, the encodings of rhombic DGGS must first be decoded into the encodings of hexagonal DGGS, which are, in turn, decoded into spatial coordinates.

From the above analysis, a comparison between the coding method proposed in this paper and the three commonly used coding methods can be drawn, as shown in Table 4. The HQBS coding method and the PYXIS coding method have good performance when performing operations such as topology query, neighborhood query, and spatial coordinate conversion based on DGGS coding in the CPU. However, both PYXIS and HQBS are encoded based on hierarchical and neighborhood relationships between hexagons, which are still irregular data for the GPU. In addition, for a highly parallelized GPU, constant table lookups and logical judgments inevitably affect the computational efficiency. Although the encoding method proposed in hierarchical grid conversion can convert hexagonal DGGS data into regular rhombic textures, it needs to record additional vertex information, which increases the amount of data when passing the DGGS data into the GPU, and it needs to be decoded twice in the GPU and will inevitably affect the efficiency of the conversion from raster data to the DGGS. Compared with these encoding methods, the DGGS encoding method in this paper is simpler. Although, to a certain extent, it reduces the efficiency of encoding conversion and indexing in the CPU, this encoding method essentially involves no logical judgment, no additional encoding calculation tables, and no additional conversions. Operations such as indexing, topological querying, and the hierarchical querying of hexagonal cells can be performed directly by coding calculations. Therefore, this method enjoys higher decoding and topology query efficiencies in the GPU, which make the method proposed in this paper for converting raster data to the hexagonal DGGS more efficient compared to other methods.

Table 4. Comparison of coding and decoding methods.

Coding Method	A Large Number of Logical Judgments	Requires Calculation Table	Requires Multiple Decodes
PYXIS	√	√	×
HQBS	×	√	×
Hierarchical grid conversion	×	×	√
Proposed in this paper	×	×	×

6.2. Impact of the Coding Method Proposed in this Paper on Bandwidth and Raster Integration Efficiency

In this paper, the rhombic tile structure is used as a texture to realize the transmission of DGGS data from the CPU to the GPU, and the coding and decoding mechanisms are specially designed for this structure. Specifically, the hexagonal cells and the rhombic tiles are encoded in the CPU, and then only the encoding is transferred to the GPU, where the data are decoded, thus reducing the PCI-e bandwidth consumption during data transfer. Assume that the concurrency of data transmission is 200 requests per second and that the data volume includes all the rhombic tiles. Figure 20 illustrates the comparison of data transmission bandwidth requirements with and without the rhombic tile structure in the data organization phase of the hexagonal DGGS. Figure 21 illustrates the overall efficiency comparison with and without the proposed encoding and decoding methods.

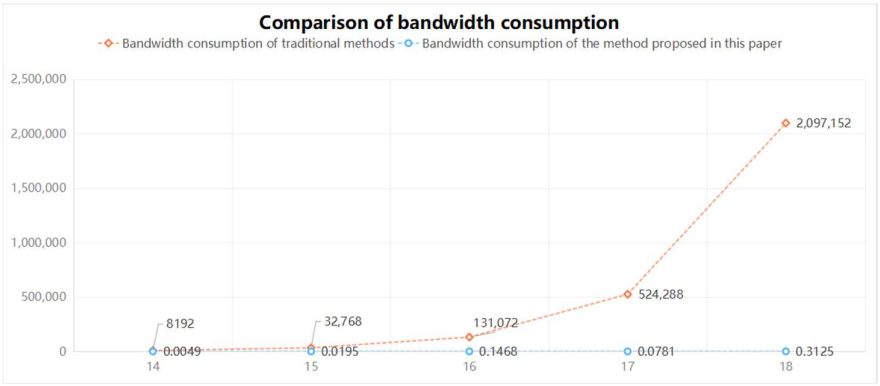

Figure 20. Comparison of bandwidth consumption.

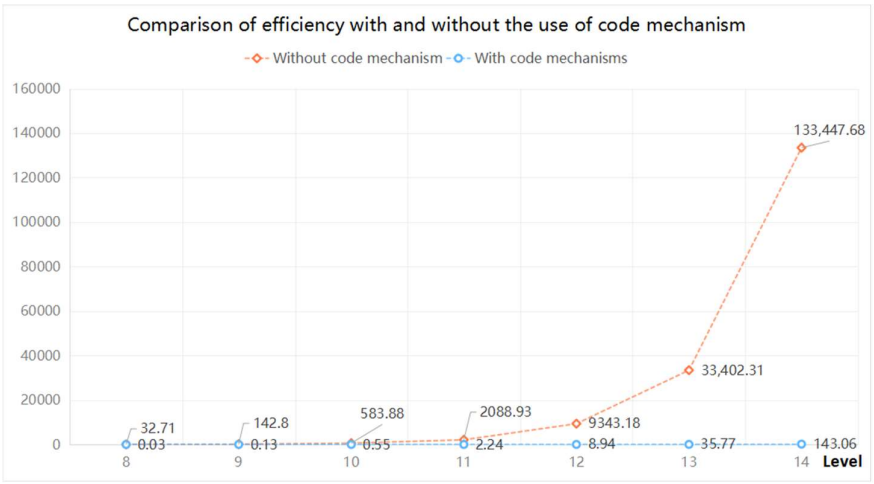

Figure 21. Comparison of efficiency with and without the use of encoding and decoding mechanisms.

6.3. Scalability of the Proposed Method

Although the method proposed in this paper is only used in the discussion and validation of the hexagonal grid with the ideal topological properties, the scheme is also applicable to grids of other topological types, based on the generation of polyhedra composed of rhombic surfaces, such as triangles, rhombuses, etc. Take the icosahedron DGGS at level 2 as an example: the cells of different topological types on a basic rhombic surface are arranged as shown in Figure 22. From the figure, it can be seen that both triangles, rhombuses, and hexagons are uniformly distributed on the polyhedron composed of rhombic faces, so all of them can be *q*2*di* coded based on the rhombic faces, and based on this, *q*2*di* coding can be converted into the coding method proposed in this paper. The cells of

the DGGS are expressed based on the coding in this paper, so, when the encoding system is consistent, it is only necessary to adjust the query method of adjacency according to the topological type of the cells. Raster data integration based on DGGSs can be achieved using the method outlined in this paper.

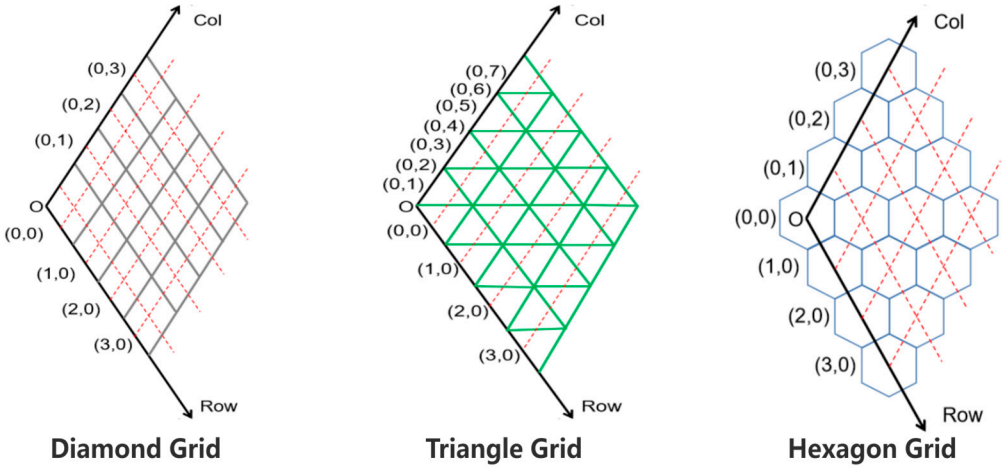

Figure 22. Various types of hexagonal cells on the surface of an icosahedron. (The black line represents the *q*2*di* coordinate system, the gray line represents rhombic cells, the green line represents triangular cells, the blue line represents hexagonal cells, and the red dashed line represents the projection of the center point of the cell into the *q*2*di* coordinate system).

To prove this conclusion, we conducted experiments with global Aster 30 m digital elevation data, and the experimental method consisted of applying the raster data integration method proposed in this paper to an icosahedron-based triangle DGGS and an icosahedron-based rhombus DGGS. The level of the triangle DGGS and the rhombus DGGS was 18. Since the number of cells in the triangle DGGS was twice as large as the number of cells in the rhombus DGGS at the same level, the raster data integration experiments on the triangle DGGS only used cells and raster data with longitudes ranging from -180° to 0°. Finally, we compared the experimental results using an icosahedron-based hexagonal DGGS, as shown in Table 5. Compared to hexagonal cells, rhombic cells have simpler adjacency relationships and, thus, are more efficient in raster data integration. Although the adjacency relationship of triangle cells is simpler than that of rhombus cells, there are two topological types of upper and lower triangles in a triangle DGGS, which have different methods of calculating the adjacency relationship. Therefore, decoding tiles with internal triangular cells in the GPU must constantly make logical judgments, which affects the efficiency of decoding.

Table 5. Comparison of integration efficiency between DGGSs with different topology types.

Topological Type	Level	Number of Cells	Time Consumption (s)
Triangle	17	687,194,767,360	158.81
Rhombus	18	687,194,767,360	134.57
Hexagon	18	687,194,767,362	143.06

7. Conclusions

In this paper, we proposed an algorithm for converting raster data to the hexagonal DGGS in the GPU based on a specially designed encoding and decoding mechanism. We first transformed the hexagonal DGGS problem into a kind of regular lattice by constructing an hexagonal DGGS data structure based on rhombic tiles, on the basis of which we designed the data organization scheme and the encoding scheme, and we managed to

reduce bandwidth occupation during data IO through the use of encoding and decoding mechanisms. Then, an asynchronous collaborative design between the CPU and the GPU was executed, aiming to optimize the computational performance of the GPU. The algorithm proved to effectively improve the efficiency of the conversion from raster data to hexagonal DGGS data.

In conclusion, the method proposed in this research has proven to be capable of implementing fast and large-scale high-resolution raster integration based on a hexagonal grid system. It has demonstrated considerable advantages in solving the problem of insufficient computational power and low efficiency of DGGSs in processing high-resolution raster data. The authors hope that this study may provide a solution to the problem of the real-time integration of high-resolution raster data based on DGGSs in the field of GIS.

The limitation of these research results is that only DGGSs based on polyhedrons such as icosahedrons and octahedrons, which are composed of diamond or triangle faces, are supported. DGGSs based on polyhedrons such as hexahedrons and dodecahedrons, which are composed of other geometric figures, are not supported.

Author Contributions: Conceptualization, L.Z. and G.L.; methodology, L.Z. and S.Z.; software, S.Z.; data curation, C.L.; writing—original draft preparation, S.Z.; writing—review and editing, L.Z.; visualization, S.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the National Natural Science Foundation of China [42076203].

Data Availability Statement: The data and codes can be explored on figshare: https://figshare.com/s/4bf94da4eaedbdd2c014 (accessed on 3 June 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

In order to more clearly represent the logic of encoding and decoding hexagonal cells using the encoding method proposed in this paper, a pseudo-code representation is given here.

```
typedef struct DGGSCell
{
        //basic rhombic surface
        int Q;
        //row and column
        int R;
        int C;
}

typedef struct GridTile
{
        //basic rhombic surface
        int Q;
        //level of the tile
        int L;
        //coding range of grid cells inside the tile
        int CellRMin;
        int CellRMax;
        int CellCMin;
        int CellCMax;
        DGGSCell* cells;
}
```

Data structure design

```
Coding of grid cells inside tiles
Go through each tile and code the grid cells according to the level and coding range of the tile
#define ThresholdValue M
    For each GridTile
        If(GridTile.L≤M)
            GridTile.code=GridTile.Q+L
        Else
            GridTile.code=GridTile.Q+L+S
            Calculate the code range inside the tile base on the code of GridTile
            For each cell in GridTile
                cell.Q=GridTile.Q
                cell.R=GridTile.CellRMin
                cell.C=GridTile.CellCMin
            EndFor
        Endif
    Endfor
    Output the relative coding of the grid cells
```

Pseudo-code

Figure A1. The algorithm for encoding.

Decoding of tile data
The rhombic tile encoding is transmitted from the CPU to the GPU, which receives it and traverses all the encodings and decodes them into *q2di* codes
For each Tilecode
GridTile.Q=The first digit of the Tilecode
GridTile.L=The second digit of the Tilecode
Calculate the code range inside the tile according to Equation (1)
For each cell in GridTile
$cell_{q2di}.Q$=GridTile.Q
$cell_{q2di}.R$+=GridTile.CellRMin
$cell_{q2di}.C$+=GridTile.CellCMin
Endfor
Endfor
Return the array consisting of *q2di* codes

Figure A2. Decoding algorithm for rhombic tile data.

References

1. Sahr, K.; White, D.; Kimerling, A.J. Geodesic Discrete Global Grid Systems. *Cartogr. Geogr. Inf. Sci.* **2013**, *30*, 121–134. [CrossRef]
2. Goodchild, M.F. Discrete Global Grids: Retrospect and prospect. *Geogr. Geo-Inf. Sci.* **2012**, *28*, 1–6.
3. Zhou, C.; Ou, Y.; Ma, T. Progresses of geographical grid systems researches. *Prog. Geogr.* **2009**, *28*, 657–662. (In Chinese)
4. Wang, L.; Ai, T.; Burghardt, D.; Shen, Y.; Yang, M. A hexagon-based method for polygon generalization using morphological operators. *Int. J. Geogr. Inf. Sci.* **2023**, *37*, 88–117. [CrossRef]
5. Wang, R.; Ben, J.; Zhou, J.; Zheng, M. Indexing Mixed Aperture Icosahedral Hexagonal Discrete Global Grid Systems. *Int. J. Geo-Inf.* **2020**, *9*, 171. [CrossRef]
6. Li, M.; McGrath, H.; Stefanakis, E. Multi-resolution topographic analysis in hexagonal Discrete Global Grid Systems. *Int. J. Appl. Earth Obs. Geoinf.* **2022**, *113*, 102985. [CrossRef]
7. Rawson, A.; Sabeur, Z.; Brito, M. Intelligent geospatial maritime risk analytics using the Discrete Global Grid System. *Big Earth Data* **2021**, *6*, 295–322. [CrossRef]
8. Ji, C.; Li, Y.; Qiu, W.; Awada, U.; Li, K. Big Data Processing in Cloud Computing Environments. In Proceedings of the 2012 International Symposium on Pervasive Systems. Algorithms, and Networks, I-SPAN 2012, San Marcos, TA, USA, 13–15 December 2012; pp. 17–23.
9. Keckler, S.W.; Dally, W.J.; Khailany, B.; Garland, M.; Glasco, D. GPUs and the future of parallel computing. *IEEE Micro* **2011**, *31*, 7–17. [CrossRef]
10. Stojanovic, N.; Stojanovic, D. High performance processing and analysis of geospatial data using CUDA on GPU. *Adv. Electr. Comput. Eng.* **2014**, *14*, 109–114. [CrossRef]
11. Mahdavi-Amiri, A.; Alderson, T.; Samavati, F. A Survey of Digital Earth. *Comput. Graph.* **2015**, *53*, 96–117. [CrossRef]
12. Tong, X.; Ben, J.; Qing, Z.; Zhang, Y. The hexagonal discrete global grid system appropriate for remote sensing spatial data. In Proceedings of the Geoinformatics 2008 and Joint Conference on GIS and Built Environment: Advanced Spatial Data Models and Analyses, Guangzhou, China, 10 November 2008.
13. Vince, A.; Zheng, X. Arithmetic and Fourier transform for the PYXIS multi-resolution digital Earth model. *Int. J. Digit. Earth* **2009**, *2*, 59–79. [CrossRef]
14. White, D. Global grids from recursive diamond subdivisions of the surface of an octahedron or icosahedrons. *Environ. Monit. Assess.* **2000**, *64*, 93–103. [CrossRef]
15. Tong, X.; Ben, J.; Wang, Y.; Zhang, Y.; Pei, T. Efficient encoding and spatial operation scheme for aperture 4 hexagonal discrete global grid system. *Int. J. Geogr. Inf. Sci.* **2013**, *27*, 898–921. [CrossRef]
16. Ben, J.; Li, Y.; Zhou, C.; Wang, R.; Du, L. Algebraic encoding scheme for aperture 3 hexagonal discrete global grid system. *Sci. China Earth Sci.* **2018**, *61*, 215–227. [CrossRef]
17. Mahdavi-Amiri, A.; Alderson, Y.; Samavati, F. Geospatial Data Organization Methods with Emphasis on Aperture-3 Hexagonal Discrete Global Grid Systems. *Cartographica* **2019**, *54*, 30–50. [CrossRef]
18. Mahdavi-Amiri, A.; Harrison, E.; Samavati, F. Hierarchical grid conversion. *Comput.-Aided Des.* **2016**, *79*, 12–26. [CrossRef]
19. Wang, J.; Shi, Y.; Qin, Z.; Chen, Y.; Cao, Z. A three-dimensional buffer analysis method based on the 3D Discrete Global Grid System. *Int. J. Geo-Inf.* **2021**, *10*, 520. [CrossRef]

20. Mahdavi-Amiri, A.; Harrison, E.; Samavati, F. Hexagonal Connectivity Maps for Digital Earth. *Int. J. Digit. Earth* **2014**, *8*, 750–769. [CrossRef]
21. Tan, L.; Wan, G.; Li, F.; Chen, X.; Du, W. GPU based contouring method on grid DEM data. *Comput. Geosci.* **2017**, *105*, 129–138. [CrossRef]
22. Lubbe, R.; Xu, W.J.; Wilke, D.N.; Pizette, P.; Govender, N. Analysis of parallel spatial partitioning algorithms for GPU based DEM. *Comput. Geotech.* **2020**, *125*, 103708. [CrossRef]
23. Kamewar, A.S. Processing geospatial images using GPU. In Proceedings of the 2017 International Conference on Emerging Trends & Innovation in ICT (ICEI), Pune, India, 3–5 February 2017; pp. 27–32.
24. Lu, M.; Wang, J.Y.; Lu, G.; Tao, W.D.; Wang, J.C. Research of raster data spatial analysis under CPU/GPU heterogeneous hybrid parallel environment-take terrain factors analysis as an example. *Comput. Eng. Appl.* **2017**, *53*, 172–177.
25. Lin, B. Research on the parallel computing and novel methodology of marine data visualization linear integral convolution algorithm based on GPU. In Proceedings of the 2015 Conference on Informatization in Education, Management and Business (IEMB-15), Guangzhou, China, 12–13 September 2015; Volume 20, pp. 96–101.
26. Stojanović, N.; Stojanović, D. Performance Improvement of viewshed analysis using GPU. In Proceedings of the International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services, Nis, Serbia, 16–19 October 2013; Volume 1, pp. 397–400.
27. Sherlock, M.J.; Hasan, F.; Samavati, F. Interactive data styling and multifocal visualization for a multigrid web-based Digital Earth. *Int. J. Digit. Earth* **2020**, *14*, 288–310. [CrossRef]
28. Yao, X.; Mokbel, M.F.; Ye, S.; Li, G.; Alarabi, L.; Eldawy, A.; Zhao, Z.; Zhao, L.; Zhu, D. LandQ^{v2}: A MapReduce-Based System for Processing Arable Land Quality Big Data. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 271. [CrossRef]
29. STCL (discreteglobalgrids.org). Available online: https://www.discreteglobalgrids.org/software/ (accessed on 3 June 2024).
30. Stough, T.; Cressie, N.; Kang, E.L.; Michalak, A.M.; Sahr, K. Spatial analysis and visualization of global data on multi resolution hexagonal grids. *Jpn. J. Stat. Data Sci.* **2020**, *3*, 107–128. [CrossRef]
31. Snyder, J.P. An Equal-area Map Projection for Polyhedral Globes. *Cartographica* **1992**, *29*, 10–21. [CrossRef]
32. Middleton, L.; Sivaswamy, J. *Hexagonal Image Processing: A Practical Approach*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006.