3D Segmented ODETLAP Compression

Wenli Li
Rensselaer Polytechnic Institute
Troy, NY, USA
liw9@rpi.edu

W. Randolph Franklin
Rensselaer Polytechnic Institute
Troy, NY, USA
mail@wrfranklin.org

Salles V. G. Magalhães
Rensselaer Polytechnic Institute
Troy, NY, USA
vianas2@rpi.edu

Marcus V. A. Andrade
Universidade Federal de Viçosa
Viçosa, MG, Brazil
marcus.ufv@gmail.com

David L. Hedin
Rensselaer Polytechnic Institute
Troy, NY, USA
hedind@rpi.edu

ABSTRACT

We propose a segmented ODETLAP compression algorithm to increase speed, and show that it is usually better in the maximum absolute error than JP3D for 3D datasets. Overdetermined Laplacian partial differential equations (ODETLAP) is a spatial approximation and data compression method. We use the CUSP library to accelerate ODETLAP approximation and the speedup is about 7 times on a GPU over a CPU core. Segmented ODETLAP compression is faster and uses less memory than unsegmented ODETLAP compression. We use the algorithm to compress several atmospheric datasets and an MRI dataset. For evaluation, we also compress the datasets using JPEG 2000 Part 10 JP3D. The results show that the compressed size of the algorithm is about 60% that of JP3D for the same maximum absolute error.

Categories and Subject Descriptors

G.1.2 [**Numerical Analysis**]: Approximation—*Least squares approximation*

Keywords

3D, compression, segmentation

1. INTRODUCTION

Overdetermined Laplacian partial differential equations (ODETLAP) is a spatial approximation and data compression method invented by Franklin [1, 2, 3]. ODETLAP has two components: approximation and lossy compression.

ODETLAP approximation computes a value for each point of a regular grid from a set of known points and their values by solving an overdetermined system of linear equations. The system includes an averaging equation for each grid point and a known-value equation for each known point. The basic form of the averaging equation is the finite-difference approximation of the Laplace's equation, which states that the approximate value of a grid point is equal to the average approximate value of adjacent grid points.

$$u(x-1,y,z)+u(x+1,y,z)+u(x,y-1,z)+u(x,y+1,z)+ u(x,y,z-1)+u(x,y,z+1)-6u(x,y,z)=0,$$

where u is the unknown approximation and each $u(x,y,z)$ is an unknown variable (in 3D). Multiplying both sides of the equation with a positive parameter R gives

$$Ru(x-1,y,z)+Ru(x+1,y,z)+Ru(x,y-1,z)+Ru(x,y+1,z)+ Ru(x,y,z-1)+Ru(x,y,z+1)-6Ru(x,y,z)=0. \quad (1)$$

R is called the smoothing factor of ODETLAP approximation. It does not change the equation but changes the weight of the equation in an overdetermined system. The known-value equation of a known point states that its approximate value is equal to its value.

$$u(x,y,z)=v(x,y,z), \quad (2)$$

where v is a value function, for example, a dataset. The system of Equations 1 and 2 has more equations than unknown variables, whose least-squares solution consists of the approximate value of each grid point. As a result, the approximate value of each, especially unknown, point is almost the average approximate value of adjacent points, and the approximate value of each known point is almost its value. R balances a trade-off between the smoothness of the approximation and its accuracy at known points by specifying the weight of averaging equations relative to known-value equations.

More formally, given a regular grid $G_{n\times n\times n}$, a set of known points $(x_i,y_i,z_i)\in G$, $i=1,\ldots,k$, and their values $v(x_i,y_i,z_i)$, $i=1,\ldots,k$, ODETLAP approximation computes an approximate value $u(x,y,z)$ for each point $(x,y,z)\in G$ by solving an overdetermined system of Equations 1 and 2. For Equation 1, a boundary point has less than 6 adjacent points within G: a face point has 5 adjacent points in G; an edge point has 4 adjacent points in G; and a corner point has 3 adjacent points in G. In this paper, we use a Neumann boundary condition of 0, which specifies that a boundary point has the same approximate value as adjacent points on

the outside of G. As a result, the Equation 1 of a boundary point has fewer terms. For example, the Equation 1 of a corner point could be

$$Ru(x-1,y,z)+Ru(x,y-1,z)+Ru(x,y,z-1)-3Ru(x,y,z)=0.$$

The system has n^3+k equations and n^3 unknown variables:

$$Ax=\begin{pmatrix}A_1\\A_2\end{pmatrix}x=\begin{pmatrix}0\\v\end{pmatrix}=b.$$

The Equation 1 coefficient matrix A_1 is $n^3\times n^3$ with at most 7 nonzero elements on each row, and the Equation 2 coefficient matrix A_2 is $k\times n^3$ with 1 nonzero element on each row. $x=(u(1,1,1),\ldots,u(n,1,1),u(1,2,1),\ldots,u(n,n,n))^T$ is the vector of unknown variables, and $v=(v(x_1,y_1,z_1),\ldots,v(x_k,y_k,z_k))^T$ is the vector of known values. The direct method to solve $Ax=b$ is to compute $x=(A^TA)^{-1}A^Tb$. A and A^TA are sparse but $(A^TA)^{-1}$ is dense and would take too much memory. Therefore, we solve $A^TAx=A^Tb$ using an iterative solver, where A^TA is a sparse $n^3\times n^3$ matrix.

ODETLAP compression selects a set of known points and values K from a regular point dataset as lossy compression, and decompresses the dataset as an ODETLAP approximation from K [4, 5, 6]. The objective is to minimize the maximum absolute error of the approximation. K is compressed using other data compression methods to reduce the compressed size of the dataset.

Algorithm 1 shows the original algorithm of ODETLAP compression. Given a point dataset, it first selects an initial set of known points, for example, random data points, and computes an approximation from the known points. While the approximation is not accurate enough, it adds a number of (unknown) data points with large errors to the set of known points, and computes a new approximation from the known points. It imposes a minimum distance between data points added in the same iteration to prevent known points from clustering.

Algorithm 1: Original ODETLAP compression

Input: A point dataset
Output: A set of known points
select an initial set of known points;
compute an ODETLAP approximation from the known points;
while *the approximation is not accurate enough* **do**
    add a number of data points with large errors to the set of known points;
    compute an ODETLAP approximation from the known points;
end

The set of known points and their values are compressed separately. The known points are represented as a binary image the size of the dataset, with 1 denoting known and 0 denoting unknown. The binary image is encoded in run-length encoding (RLE) and each run length $l<65536$ is stored in this format: if $l<254$, a byte for l; if $254\le l<510$, a marker byte `0xFE` and a byte for $l-254$; if $510\le l<766$, a marker byte `0xFF` and a byte for $l-510$; and if $l>766$, two marker bytes `0xFFFF` and two bytes for l. The values are encoded in delta encoding and compressed by the bzip2 program.

The maximum absolute error is important in some applications. For example, a large error in the elevation data indicates a potential hazard for a flying aircraft, a ship, or a submarine, while the average error is irrelevant. There is a similar concern with any collision detection problem, e.g., in robotics and mechanical assembly. ODETLAP approximation is reasonably fast, especially with GPU acceleration. However, ODETLAP compression is slow because it may involve hundreds of thousands of ODETLAP approximations, depending on the size of the dataset. We can either reduce the number of approximations or reduce the size of each approximation to increase the speed.

As the inspiration of this work, Mitášová and Mitáš [7, 8] developed a segmentation procedure for the interpolation of large datasets using completely regularized splines. The idea is based on the local behavior of the interpolation function. The procedure divides a regular grid into square segments such that the number of data points in the 3×3 neighborhood of each segment is less than a threshold. Interpolated values in each segment are computed from data points in its 3×3, 5×5, or larger neighborhood such that the number of data points is larger than a second threshold. It improves the smoothness of the interpolation across segment boundaries. Segmentation reduces both computation time and memory requirement. JPEG [9] and JPEG 2000 [10] split an image into blocks or tiles and transform and encode each block separately. Tiling reduces memory requirement but may produce a blocking artifact.

Tiling was also used to reduce the complexity of ODETLAP approximation, and overlapping tiles were used to enhance smoothness. Stookey et al. [11, 12] proposed a parallel ODETLAP approximation algorithm for terrain approximation on an IBM Blue Gene/L. The algorithm divides a grid into overlapping patches and computes an ODETLAP approximation for each patch, then merges the results using bilinear interpolation. For example, it divides the grid into patches of $w\times w$ points, whose lower left corners are at $(iw/2,jw/2)$, $i,j=0,1,\ldots$, so that each point is in at most four patches. The patches are grouped into blocks, and the blocks are approximated and interpolated in parallel. The algorithm can be used in ODETLAP compression.

Li et al. [13, 14, 15] proposed an ODETLAP compression algorithm for 3D and 4D oceanographic data compression and had better results than the 3D set partitioning in hierarchical trees (3D SPIHT) of Said, Kim, and Pearlman [16, 17]. The algorithm uses an ODETLAP approximation algorithm that divides a grid into two interleaved meshes of boxes, computes an ODETLAP approximation for each box, and merges the results by weighted average. For example, it divides the grid into a mesh of boxes whose vertices are at (iw,jw,kw), $i,j,k=0,1,\ldots$, and another mesh of boxes whose vertices are at $(iw+w/2,jw+w/2,kw+w/2)$, $i,j,k=0,1,\ldots$, so that each point is in at most two boxes. The approximation is accelerated on the GPU using an iterative solver from the CUSP library, while the rest of the algorithm is implemented in MATLAB. Benedetti et al. [18, 19] accelerated ODETLAP approximation and compression in C++

on the GPU using the CUSP and Thrust libraries.

In this paper, we propose a segmented ODETLAP compression algorithm to increase the speed by reducing the size of each approximation. The algorithm is fundamentally different from the patched algorithm of Stookey et al. and the boxed algorithm of Li et al. because it does not approximate the entire dataset each time and does not have a separate merging step. We use the algorithm to compress 3D datasets and evaluate its relative performance to JP3D. Results show that the algorithm is usually better in the maximum absolute error. First, we renew the implementation of GPU-accelerated ODELTAP approximation in Section 2.

2. GPU-ACCELERATED ODETLAP APPROXIMATION

GPUs are massively parallel devices containing thousands of processing units. They were designed for computer graphics applications but are fully programmable and optimized for SIMD vector operations. Using GPUs for scientific computing is called general-purpose computing on graphics processing units (GPGPU) [20]. Although a CPU core is much more powerful than a GPU core, a GPU has many more cores than a CPU. Latest GPUs have up to a few hundred GB/s of memory bandwidth and a few TFLOPS of single-precision processing power, but theoretical peak performance is very hard to achieve.

Compute unified device architecture (CUDA) [21] is a parallel computing platform and programming model for NVIDIA GPUs. While CUDA C/C++ or CUDA Fortran is the standard API for CUDA-enabled GPUs, a GPU-accelerated library [22] is often a better option. For example, Thrust [23] is a C++ template library that provides a high-level interface for CUDA programming. Thrust provides two generic containers, one in host (CPU) memory and one in device (GPU) memory, and multiple algorithms for transformations, reductions, prefix-sums, reordering and sorting. Built on Thrust, CUSP [24] is a C++ template library for sparse linear algebra. CUSP provides multiple sparse matrix formats, iterative solvers and preconditioners.

Solving large overdetermined systems is time-consuming and benefits from GPU acceleration. GPU-accelerated ODETLAP approximation evolves as hardware and software technologies develop. Our implementation of GPU-accelerated ODETLAP approximation using the CUSP library has three parts:

1. build the coefficient matrix A and right-hand side b of the overdetermined system on the CPU;
2. compute A^T, A^TA, and A^Tb on the GPU;
3. solve $A^TAx = A^Tb$ on the GPU.

The workstation used in this paper has two Intel Xeon E5-2687W CPUs and one NVIDIA Tesla K20Xm GPU accelerator, running Ubuntu 16.04 LTS. The value type of the data structures is the single precision float type and the relative tolerance of the iterative solver is 1×10^{-6}. The GPU has much higher single precision processing power than double precision processing power. The dataset is a $360 \times 180 \times 24$ atmospheric dataset 'CH4_VMR', described in Section 4, with 1% random nonempty data points selected as known points. The running time of ODETLAP approximation is measured as the average time of 20 runs with the smoothing factor R = 1.

Table 1: The running time and speedup of GPU-accelerated ODETLAP approximation. CPU time: the running time on a CPU core in seconds. GPU time: the running time on the GPU in seconds. Speedup: CPU time over GPU time.

	CPU time	GPU time	Speedup
Part 1	0.15	–	–
Part 2	1.75	0.62	2.84
Part 3	42.52	5.65	7.53

CUSP has five sparse matrix formats: coordinate (COO), compressed sparse row (CSR), diagonal (DIA), ELLPACK/ITPACK (ELL), and hybrid ELL/COO (HYB). We use the COO format for host A, device A, and device A^T, because it is fast for basic operations. The format of device A^TA affects the performance of the iterative solver, whose main operation is sparse-matrix-vector multiplication. We tried each format for device A^TA using the conjugate gradient method and found that the ELL or HYB format is the fastest.

CUSP has numerous iterative solvers. Relaxation methods include Gauss-Seidel, Jacobi, and successive over-relaxation (SOR). Krylov subspace methods include biconjugate gradient (BiCG), biconjugate gradient stabilized (BiCGstab), conjugate gradient (CG), conjugate residual (CR), and generalized minimum residual (GMRES). We tried each method using the ELL format for device A^TA and found that the Jacobi, BiCGstab, and CR methods do not converge; the Gauss-Seidel and SOR methods are very slow and the GMRES method is slow; and the CG method is faster than the BiCG method. CUSP has smoothed aggregation-based algebraic multigrid (AMG), approximate inverse (AINV), and diagonal preconditioners. Using the CG method, the AINV and diagonal preconditioners do not reduce the running time. The AMG preconditioner significantly reduces the running time but uses more memory. In summary, we use the ELL format for device A^TA, the CG method, and the AMG preconditioner.

Using the CUSP library, host data structures are computed on the CPU and device data structures are computed on the GPU. Storing all data structures in host memory puts all computation on the CPU, so that we can run the same program on a CPU core. Table 1 shows the running time of each part of the program on a CPU core and on the GPU, and the GPU/CPU speedup. Part 1 is only run on the CPU. Part 1 and part 2 use little time compared to part 3. The overall speedup of the program is about 7 times.

3. SEGMENTED ODETLAP COMPRESSION

ODETLAP compression can be accelerated by dividing a dataset into segments and compressing each segment separately, like JPEG and JPEG 2000, because the running time of ODETLAP approximation grows fast as the size of the grid increases. In addition, segmentation reduces the memory requirement of ODETLAP approximation. However, compressing segments separately produces blocking arti-

facts across segment boundaries in the decompressed dataset, which consists of an ODETLAP approximation in each segment, computed from the known points in the segment. To improve smoothness and reduce artifacts, the approximation in each segment can be computed as the center of a larger approximation in a neighborhood of the segment, using the known points in the neighborhood (including the segment), like Mitášová and Mitáš [7, 8]. The idea is based on the local behavior of ODETLAP approximation: the approximate value of a grid point is mostly determined by the values of the nearest known points. If there are sufficient known points in the neighborhood and outside of the segment, the value of a point in the segment approximation is close to its value in an ODETLAP approximation of the dataset from the known points in all segments.

The main idea of the segmented ODETLAP compression algorithm is to divide a dataset into segments and compress each segment in association with adjacent segments. The algorithm selects a set of known points in each segment so that the maximum absolute error (MAXE) of the segment approximation is not larger than a target threshold. The segment approximation is computed as the center of an ODETLAP approximation in its neighborhood. The algorithm adds known points in the segments in a round-robin fashion so as to compress them in synchrony. As Algorithm 2 shows, given a segmented point dataset, the algorithm first selects an initial set of known points and marks all segments as needing processing, which means the MAXE of the segment approximation may be larger than the target. While there are segments that need processing, and for each such segment in random order, the algorithm computes an approximation in the segment, and if the MAXE of the approximation is larger than the target, it adds a known point in the segment, otherwise it marks the segment as not needing processing. The approximation in a segment changes and thus the segment needs processing if a known point is added in its neighborhood. Therefore, if the algorithm adds a known point in a segment, it marks all other segments whose neighborhood contains the point as needing processing. The algorithm terminates after the MAXE of each segment approximation is not larger than the target, so that the MAXE of the dataset approximation is not larger than the target. As datasets have different ranges, we will specify error metrics and the target relative to the data range.

The parameters of the algorithm are:

- R: the smoothing factor of ODETLAP approximation, which is 0.01 in the rest of the paper;
- segment width (SW): the width of a segment; each segment is SW×SW×SW (in 3D);
- outer width (OW): the width between a segment boundary and its neighborhood boundary; the neighborhood is (SW+2OW)×(SW+2OW)×(SW+2OW);
- target: the target MAXE of each segment approximation.

Let n be the size of the grid (dataset). The time complexity of the conjugate gradient method is $O(n^{4/3})$ for three-dimensional elliptic problems [25]. With segmentation, the size of the grid (neighborhood) is $(\mathrm{SW}+2\mathrm{OW})^3$, and the time complexity of the method is $O((\mathrm{SW}+2\mathrm{OW})^4)$. However, as the neighborhood size decreases, the speedup of GPU-accelerated ODETLAP approximation may also decrease, because the GPU needs a high occupancy to be efficient.

Algorithm 2: Segmented ODETLAP compression

```
Input: A segmented point dataset
Output: A set of known points and values
select an initial set of known points;
mark all segments as needing processing;
while there are segments that need processing do
    foreach segment s that needs processing do
        compute an approximation in s as the center of an
        ODETLAP approximation from the known points in
        its neighborhood;
        if the MAXE of the approximation is larger than a
        target then
            add the data point p ∈ s with the largest error to
            the set of known points;
            foreach segment t, t ≠ s, intersecting the
            neighborhood of s do
                if p is in the neighborhood of t then
                    mark t as needing processing;
                end
            end
        else
            mark s as not needing processing;
        end
    end
end
```

A dataset may have empty data points that do not have a value, which is often indicated by a special value not in the data range. A dataset may also have many data points that have the same value. In the first case, the values of empty data points can be interpolated for ODETLAP compression, but it is impossible to distinguish between empty and nonempty points in the decompressed dataset. In the second case, it is less efficient to select many known points of the same value. The solution is to include a nonempty mask of the dataset in its compression. The nonempty mask is a binary image the size of the dataset, with 1 denoting nonempty and 0 denoting empty. An empty value is also included in the compression, which is either the special value of empty points or a common value in the dataset. The empty value is assigned to the empty points in the dataset approximation. With the nonempty mask, the algorithm does not select empty points as known points, and the decompressed dataset is accurate at the empty points.

Using a lower precision or fewer bits for known point values can effectively reduce the size of their compression without significantly increasing the error of the decompressed dataset (if the precision is not too low). Regardless of whether the values are integer or floating point, they can be quantized on a small range of integers and encoded in a few bits. We can quantize the values of known points either before they are selected or after they are selected. Quantizing the values after the known points are selected makes it harder to control the error of the dataset approximation. Therefore, we quantize the dataset before selecting the known points. In this paper, we quantize the dataset on integers 0–255 or the range of 8 bits. To quantize a dataset on 0–255, we find the maximum

and minimum values *vmax* and *vmin* of the dataset, and calculate $scale = 255/(vmax - vmin)$. Then we convert each value v to $\text{round}((v - vmin) \cdot scale)$. Now, the dataset approximation has a different range from the dataset. To decompress the dataset, we clamp the dataset approximation between 0 and 255, and convert each approximate value v to $v/scale + vmin$. Let the target MAXE be a percent of the data range. Quantizing the dataset on 0–255 changes its range to 255 and induces an absolute error of at most $0.5/255 = 0.196\%$ at each point. The algorithm guarantees the target MAXE of the dataset approximation from the quantized dataset. The quantized and inverse-quantized dataset has an absolute error of at most 0.196% from the dataset. Therefore, the decompressed dataset has a MAXE of at most 0.196% more than the target.

The algorithm to decompress a dataset from a nonempty mask and a set of known points and values is:

1. compute an approximation of the quantized dataset by computing an approximation for each segment;
2. clamp the dataset approximation between 0 and 255 and inverse-quantize it;
3. assign the empty value to the empty points.

The ODETLAP compression of a dataset consists of a header, a nonempty mask, a known point mask, and the known point values. The nonempty mask is compressed by a pipeline of run-length encoding (RLE), variable-length quantity (VLQ) and the ZPAQ v1.10 archiver. The RLE of the nonempty mask is the alternating run lengths of 0s and 1s, starting with a run length of 0s. VLQ is a binary format that uses one byte for an integer in 0–127, two bytes for an integer in 128–$(128^2 - 1)$, and so on. ZPAQ is an open source file archiver. The known point mask is also compressed by a pipeline of RLE, VLQ, and ZPAQ. However, the RLE of the known point mask is the run lengths of 0s between 1s, which is not in a strict sense an RLE but is more efficient if the ratio of 1s is not too large. The known point values are compressed by a pipeline of offset delta encoding, VLQ, and ZPAQ. The delta encoding of the values contains negative deltas, which can not be stored in VLQ. Offsetting the deltas makes them nonnegative: find the minimum delta *dmin*; subtract *dmin* from each delta; and prepend -*dmin* to the deltas. The header has 18 bytes:

- the numbers of slices, rows, and columns of the dataset: 2 bytes each;
- SW and OW: 1 byte each;
- *vmin* and *scale* for inverse-quantization: 4 bytes each;
- the empty value: 2 bytes.

We use regularly spaced data points as the initial known points of the algorithm, because they are evenly distributed across the dataset, and they help reduce the compressed size of the known point mask. For a given target, if the mask size is larger than the values size, using more regular initial known points often reduces the sum of the two sizes, and if the values size is larger than the mask size, quantizing the dataset on a smaller range often reduces the sum of the two sizes. If the target is large, the algorithm should use less regular initial known points, and if the target is small, the algorithm should use more regular initial known points. The optimal initial set of known points depends on the dataset and the target. In this paper, we use a fixed initial set of known points at $(4i + 2, 4j + 2, 4k + 2)$, $i, j, k = 0, 1, \ldots$, except empty data points. The interval between them is 4 in each dimension so that they are at most 1/64 of the dataset.

Table 2: Statistics of the datasets. Empty: the percentage of empty data points for the atmospheric datasets or the percentage of zero data points for the MRI dataset.

Dataset	Minimum	Maximum	Median	Empty
CH4_VMR	$1.55 \cdot 10^{-7}$	$2.02 \cdot 10^{-6}$	$1.62 \cdot 10^{-6}$	2.02%
CO_VMR	$1.52 \cdot 10^{-8}$	$3.03 \cdot 10^{-7}$	$4.07 \cdot 10^{-8}$	2.00%
GPHeight	1	50017	18204	1.99%
O3_VMR	$1.23 \cdot 10^{-8}$	$1.41 \cdot 10^{-5}$	$1.08 \cdot 10^{-6}$	1.99%
Temperature	188.0	311.9	235.5	1.99%
MRI	0	3866	0	60.59%

4. RESULTS

The datasets tested are five $360 \times 180 \times 24$ atmospheric datasets and one $256 \times 256 \times 160$ MRI dataset.

The atmospheric datasets are the January 2015 AIRX3STM [26] grid fields that have an extra dimension of 24 standard pressure levels. The basenames of the grid fields are CH4_VMR (CH4 volume mixing ratio), CO_VMR (CO volume mixing ratio), GPHeight (geopotential height), O3_VMR (O3 volume mixing ratio), and Temperature (atmospheric temperature). Each field appears in four grids, tagged _A, _D, _TqJ_A, and _TqJ_D. The grids are 360×180 with 1×1 $(^\circ)^2$ cells. For example, CH4_VMR_A and CH4_VMR_D are the same field in two grids. We only use the fields in the grid tagged _A, meaning ascending orbit and individual quality control. We use each grid field as a regular point dataset and call them by their basenames: CH4_VMR, CO_VMR, GPHeight, O3_VMR, and Temperature. Table 2 shows some statistics of the atmospheric datasets. GPHeight is integer and the other four are floating point. They all have about 2% empty data points, indicated by the special value -9999. Figure 1 shows slices 0, 8, and 16 of each of the atmospheric datasets. Empty data points are shown in white, which appear to be landmasses.

The MRI dataset is the co-registered, averaged image from four individual scan images of MR session OAS1_0001_MR1, OASIS cross-sectional MRI data [27]. The individual scan images are $256 \times 256 \times 128$ with $1 \times 1 \times 1.25$ mm^3 voxels. The averaged image is $256 \times 256 \times 160$ with $1 \times 1 \times 1$ mm^3 voxels. Table 2 shows that the MRI dataset has about 60% zero data points. Figure 2 shows slices 0, 40, 80, and 120 of the MRI dataset. Nonzero data points are in center.

We compress the datasets using the segmented ODETLAP compression algorithm. The value type of data structures is the single precision float type and the relative tolerance of the iterative solver is 1×10^{-6}. The smoothing factor R of ODETLAP approximation is 0.01. The atmospheric datasets are compressed with SW = 12 and OW = 6, so that a segment is $12 \times 12 \times 12$ and the neighborhood is $24 \times 24 \times 24$. The MRI dataset is compressed with SW = 16 and OW = 8. In general, as SW and OW increase, the running time of the algorithm increases, but the number of known points selected for a given target MAXE decreases. The algorithm selects an initial set

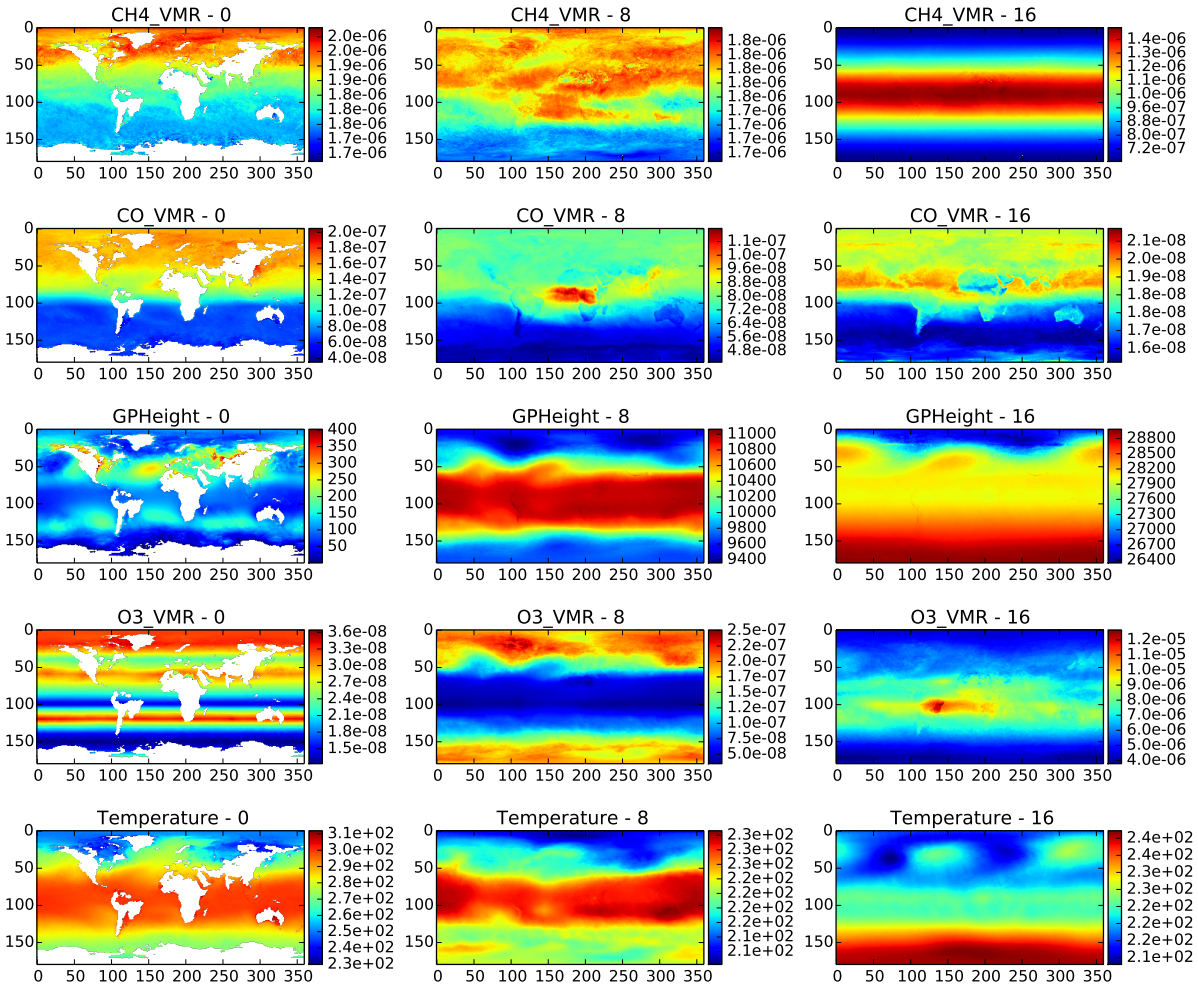

Figure 1: Slices 0, 8, and 16 of the $360 \times 180 \times 24$ atmospheric datasets.

of known points at $(4i+2, 4j+2, 4k+2)$, $i, j, k = 0, 1, \ldots$, except empty or zero data points. For the atmospheric datasets, the initial known points are about 24000 data points. For the MRI dataset, the initial known points are 64613 data points. Using more initial known points decreases the running time but increases the number of known points selected, which may or may not increase the size of the compression. Error metrics and the target are specified as percentages of the data range. We compress the atmospheric datasets with targets 3%, 2.5%, 2%, 1.5%, 1%, and 0.5%, and compress the MRI dataset with targets 6%, 5%, 4%, 3%, 2%, and 1%. In addition, we compress the datasets with the target 0%, by selecting all nonempty or nonzero data points as known points. Selecting all nonempty data points does not require the algorithm, and the compression does not have a known point mask.

Table 3 shows the results of the algorithm for each dataset and target. The results are: the number of known points selected, where all n/e means all nonempty or nonzero data points; the average absolute error (AVGE) of the decompressed dataset; the root-mean-square error (RMSE) of the decompressed dataset; the maximum absolute error (MAXE) of the decompressed dataset; the size of the compressed known point mask; the size of the compressed known point values; the total size of the compressed dataset; and the compression ratio. The compressed dataset includes the header and the compressed nonempty mask, known point mask, and known point values. The size of the nonempty mask is 2433, 2375, 2357, 2356, 2358, and 17206 bytes, respectively, for datasets CH4_VMR, CO_VMR, GPHeight, O3_VMR, Temperature, and MRI. The original size of each atmospheric dataset is 6220800 bytes (5.9 MB) and the original size of the MRI dataset is 20971520 bytes (20 MB). The results show that MAXE is at most 0.20% larger than the target. AVGE and RMSE are usually much smaller than MAXE. The number of known points and the compressed sizes increase fast as the target decreases. The size of the known point mask is 0 when all nonempty data points are selected. The results for dataset GPHeight show an anomaly: the compression ratio for 0% target is larger than the compression ratio for 0.5% target. The reason is that the size of known point values is much smaller than the size of the known point mask for the dataset. Although the size of known point values increases a lot from 0.5% to 0% target, the size of the known point mask decreases even more (to 0). Therefore, it is important to balance the two sizes.

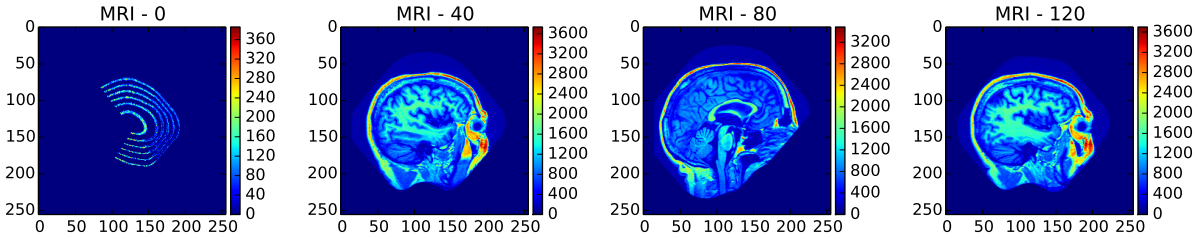

Figure 2: Slices 0, 40, 80, and 120 of the $256 \times 256 \times 160$ MRI dataset.

Table 3: Results of segmented ODETLAP compression. Target: the target MAXE for the quantized dataset. Points: the number of known points selected; all n/e means all nonempty or nonzero data points. Mask: the size of the compressed known point mask in bytes. Values: the size of the compressed known point values in bytes. Total: the total size of the header, nonempty mask, known point mask, and known point values in bytes. Ratio: the compression ratio.

Dataset	Target	Points	AVGE	RMSE	MAXE	Mask	Values	Total	Ratio
CH4_VMR	3.0%	26711	0.53%	0.74%	3.17%	2516	5829	10794	576.3
CH4_VMR	2.5%	28494	0.48%	0.65%	2.68%	3785	6478	12712	489.4
CH4_VMR	2.0%	31557	0.43%	0.57%	2.18%	5792	7470	15711	396.0
CH4_VMR	1.5%	39691	0.40%	0.51%	1.66%	10890	9359	22698	274.1
CH4_VMR	1.0%	69667	0.30%	0.38%	1.19%	27592	16022	46063	135.0
CH4_VMR	0.5%	202848	0.16%	0.20%	0.69%	80552	40360	123361	50.4
CH4_VMR	0.0%	All n/e	0.10%	0.11%	0.20%	0	210841	213290	29.2
CO_VMR	3.0%	26868	0.67%	0.90%	3.16%	2922	8038	13351	465.9
CO_VMR	2.5%	29199	0.62%	0.83%	2.69%	4834	9253	16478	377.5
CO_VMR	2.0%	34681	0.55%	0.68%	2.17%	8688	11144	22223	279.9
CO_VMR	1.5%	43567	0.46%	0.55%	1.68%	14291	14770	31452	197.8
CO_VMR	1.0%	67289	0.31%	0.37%	1.18%	27635	23477	53503	116.3
CO_VMR	0.5%	153145	0.16%	0.19%	0.69%	64784	49068	116243	53.5
CO_VMR	0.0%	All n/e	0.10%	0.11%	0.20%	0	213858	216249	28.8
GPHeight	3.0%	28813	0.59%	0.79%	3.17%	3149	1994	7516	827.7
GPHeight	2.5%	30734	0.56%	0.74%	2.66%	3726	2149	8248	754.2
GPHeight	2.0%	33642	0.50%	0.65%	2.18%	5118	2419	9910	627.7
GPHeight	1.5%	39659	0.46%	0.58%	1.68%	8134	2992	13499	460.8
GPHeight	1.0%	67466	0.39%	0.46%	1.18%	21090	5186	28649	217.1
GPHeight	0.5%	208150	0.22%	0.26%	0.69%	72046	12130	86549	71.9
GPHeight	0.0%	All n/e	0.10%	0.11%	0.20%	0	41872	44245	140.6
O3_VMR	3.0%	42321	0.69%	0.96%	3.17%	11425	11382	25179	247.1
O3_VMR	2.5%	48743	0.60%	0.82%	2.68%	14834	13753	30959	200.9
O3_VMR	2.0%	59703	0.50%	0.68%	2.19%	20161	17585	40118	155.1
O3_VMR	1.5%	81677	0.40%	0.53%	1.69%	29728	24583	56683	109.7
O3_VMR	1.0%	131809	0.28%	0.36%	1.19%	48028	38395	88795	70.1
O3_VMR	0.5%	262396	0.15%	0.19%	0.69%	80220	68702	151294	41.1
O3_VMR	0.0%	All n/e	0.10%	0.12%	0.20%	0	186058	188430	33.0
Temperature	3.0%	41074	0.95%	1.18%	3.16%	11713	12517	26604	233.8
Temperature	2.5%	50313	0.83%	1.02%	2.68%	17080	15669	35123	177.1
Temperature	2.0%	65957	0.70%	0.84%	2.18%	25683	20695	48752	127.6
Temperature	1.5%	96786	0.53%	0.64%	1.70%	40782	29797	72953	85.3
Temperature	1.0%	163326	0.36%	0.43%	1.19%	66899	45963	115236	54.0
Temperature	0.5%	347499	0.18%	0.22%	0.69%	117735	80366	200475	31.0
Temperature	0.0%	All n/e	0.10%	0.11%	0.20%	0	209563	211937	29.4
MRI	6%	292724	0.60%	1.27%	6.19%	154989	258259	430472	48.7
MRI	5%	362331	0.52%	1.09%	5.19%	186616	319131	522971	40.1
MRI	4%	471515	0.43%	0.90%	4.19%	229597	412245	659066	31.8
MRI	3%	652309	0.33%	0.69%	3.19%	288544	563325	869093	24.1
MRI	2%	982647	0.22%	0.47%	2.19%	372541	825539	1215304	17.3
MRI	1%	1755817	0.11%	0.23%	1.19%	501646	1370377	1889247	11.1
MRI	0%	All n/e	0.04%	0.07%	0.20%	0	2532834	2550058	8.2

Table 4: Smoothness measures of the datasets normalized on [0, 1], and the compression ratio for 2% target. RMSL: root-mean-square Laplacian. RMSB: root-mean-square biharmonic.

Dataset	RMSL	RMSB	Ratio - 2%
CH4_VMR	0.017	0.051	396.0
CO_VMR	0.015	0.054	279.9
GPHeight	0.018	0.044	627.7
O3_VMR	0.034	0.071	155.1
Temperature	0.032	0.077	127.6
MRI	0.089	0.400	17.3

Table 5: Interpolated compressed sizes in KB of segmented ODETLAP compression and JP3D when MAXE is 2% and 1%.

	ODETLAP		JP3D	
Dataset	2%	1%	2%	1%
CH4_VMR	17.7	73.3	71.8	192.6
CO_VMR	24.9	74.9	90.9	180.0
GPHeight	10.9	48.9	18.0	48.2
O3_VMR	45.3	110.2	71.6	152.4
Temperature	56.3	144.4	111.2	207.5
MRI	1313.8	1970.3	1556.1	1984.8

Some datasets compress better than others. For example, the compression ratio is 396.0 for dataset CH4_VMR and 127.6 for dataset Temperature when the target is 2%. We found that smoother datasets compress better. The smoothness of a dataset can be characterized by its Laplacian. Because the datasets have different ranges, we normalize them on [0, 1], and compute the root-mean-square of the Laplacian of the normalized datasets. As shown in Table 4, the value is loosely related to the compression ratios when the target is 2%, but it indicates that datasets CH4_VMR, CO_VMR, and GPHeight compress better. A better smoothness measure is the Laplacian of the Laplacian, or the biharmonic. We compute the root-mean-square of the biharmonic of the normalized datasets. As shown in Table 4, the value is closely related to the compression ratios when the target is 2%. This analysis does not take quantization into consideration.

5. EVALUATION

To evaluate the results of the algorithm, we also compress the datasets using JPEG 2000 Part 10 JP3D [28]. JP3D extends JPEG 2000 Part 1 for 3D volumetric compression. It uses a 3D version of the embedded block coding by optimized truncation (EBCOT) algorithm of JPEG 2000 Part 1. The algorithm divides a volumetric dataset into cuboid tiles and uses discrete wavelet transform (DWT) to decompose each tile into subbands. The subbands are partitioned into dyadic-sized code blocks and each code block is encoded independently. We use OpenJPEG v2.1.0, an open-source JPEG 2000 codec.

JP3D also needs a nonempty mask for datasets with empty data points, because there is no way to distinguish between empty and nonempty points in the decompression. To compress the atmospheric datasets, we first quantize them on integers 0–255, because JP3D compresses integers while the datasets are floating point. Quantizing the datasets on 0–255 produces very small errors, as shown in Table 3 for 0% target. Then we interpolate the values of empty points using nearest-neighbor interpolation, because JP3D requires a value at every point. Then we manually find the best possible compression parameters for the datasets. The size of code block is $64 \times 32 \times 8$, which has to do with the shape of the datasets. The number of resolutions in x, y, and z axis is 8, 8, and 1 (resolutions in x and y axes have to be the same), which also has to do with the shape of the datasets. The coding algorithm is 3D-EBCOT. We compress the quantized datasets with the target PSNR 40, 40.5, 41, ..., until lossless compression is achieved. The JP3D compression of an atmospheric dataset consists of a 10-byte header (*vmin* and *scale* for inverse-quantization, and the empty value), a nonempty mask, and a JP3D file. To decompress the dataset, we decompress and inverse-quantize the JP3D file, and assign the empty value to the empty points.

The process of compressing the MRI dataset is simpler because it is integer and does not have empty data points. We manually find the best possible compression parameters for the dataset. The size of code block is $64 \times 64 \times 32$, related to the shape of the dataset. The number of resolutions in x, y, and z axis is 10, 10, and 3. We compress the dataset with the target PSNR 40, 40.5, 41, ..., 99.5. The compressed dataset is a JP3D file.

Figure 3 shows the compressed size and approximation error plots of JP3D and the algorithm for each dataset and error metric. The horizontal axis is the compressed size in KB and the verical axis is an approximation error in percentage. In each plot, the blue circles are the JP3D results (some may be outside the plot). The green circles are the ODETLAP results. And the red circle is the result of compressing all nonempty or nonzero data points. The plots show that JP3D is usually better in AVGE and RMSE but ODETLAP is usually better in MAXE. The relative performance of ODETLAP to JP3D, however, is different from dataset to dataset. ODETLAP is much better in MAXE and/or closer in AVGE and RMSE for datasets CH4_VMR, CO_VMR, and MRI. It performs less favorably for datasets GPHeight, O3_VMR, and Temperature.

Table 5 shows the compressed sizes of the algorithm and JP3D when MAXE is 2% and 1% for each dataset. The sizes are linearly interpolated from the two results closest to each MAXE. The ODETLAP size is smaller than the JP3D size in most cases. It is about 60% the JP3D size in average.

6. CONCLUSIONS

We use segmentation to increase the speed of ODETLAP compression by reducing the size of each approximation in iterative selection. The algorithm can be further accelerated by reducing the number of approximations. For example, it can add multiple known points in each iteration, and use sub-segmentation to spread out the points. If the target MAXE is small, using more regular initial known points not only reduces the time but also improves the performance. Segmentation is good for capturing local trends but bad for capturing global trends, by limiting the influence of known

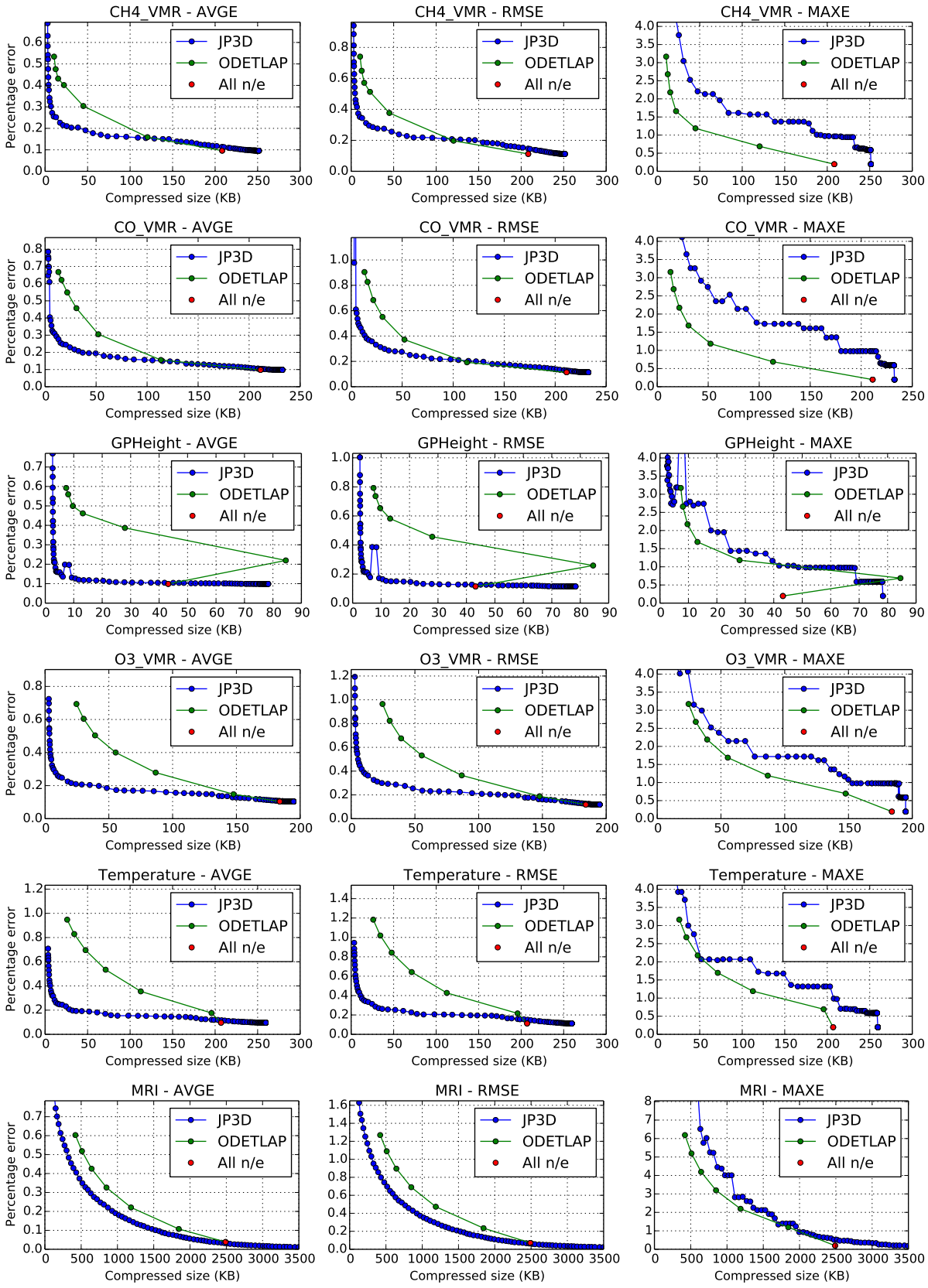

Figure 3: Compressed sizes in KB and approximation errors in percentage of JP3D and segmented ODETLAP compression. AVGE: average absolute error. RMSE: root-mean-square error. MAXE: maximum absolute error. All n/e means all nonempty data points.

points. The objective of the algorithm is to minimize the maximum absolute error. Results for 3D atmospheric and MRI datasets show that it is usually better in MAXE than JP3D. For the same MAXE, the compressed size of the algorithm is about 60% that of JP3D. In the future, we may use multi-core CPUs to accelerate both ODETLAP approximation and compression. We would like to find better ways to explore the regularity of known points. And we will use the algorithm to compress 4D datasets.

Acknowledgement

This research was partially supported by NSF grant IIS-1117277 and CAPES (Ciência sem Fronteiras).

7. REFERENCES

[1] W. Randolph Franklin and Michael Gousie. Terrain elevation data structure operations. In C. Peter Keller, editor, *Proceedings of the 19th International Cartographic Conference*, ICC '99, pages 1011–1020, August 1999.

[2] W. Randolph Franklin, Metin Inanc, and Zhongyi Xie. Two novel surface representation techniques. In *Proceedings of AutoCarto 2006*, June 2006.

[3] W. Randolph Franklin, You Li, Tsz-Yam Lau, and Peter Fox. CUDA-accelerated HD-ODETLAP: Lossy high dimensional gridded data compression. In Xuan Shi, Volodymyr Kindratenko, and Chaowei Yang, editors, *Modern Accelerator Technologies for Geographic Information Science*, pages 95–111. Springer US, Boston, MA, USA, 2013.

[4] Zhongyi Xie, W. Randolph Franklin, Barbara Cutler, Marcus A. Andrade, Metin Inanc, and Daniel M. Tracy. Surface compression using over-determined laplacian approximation. In *Proceedings of SPIE Volume 6697, Advanced Signal Processing Algorithms, Architectures, and Implementations XVII, 66970F*, September 2007.

[5] Zhongyi Xie, Marcus A. Andrade, W. Randolph Franklin, Barbara Cutler, Metin Inanc, Jonathan Muckell, and Daniel M. Tracy. Progressive transmission of lossily compressed terrain. In *Proceedings of XXXIV Conferencia Latinoamericana de Informtica*, CLEI '08, pages 8–12, September 2008.

[6] Zhongyi Xie. Representation, compression and progressive transmission of digital terrain data using over-determined laplacian partial differential equations. Master's thesis, Rensselaer Polytechnic Institute, 2008.

[7] Helena Mitášová and Lubos Mitáš. Interpolation by regularized spline with tension: I. theory and implementation. *Mathematical Geology*, 25(6):641–655, 1993.

[8] Helena Mitášová and Jaroslav Hofierka. Interpolation by regularized spline with tension: II. application to terrain modeling and surface geometry analysis. *Mathematical Geology*, 25(6):657–669, 1993.

[9] William B. Pennebaker and Joan L. Mitchell. *JPEG: Still Image Data Compression Standard*. Kluwer Academic, Norwell, MA, USA, 1st edition, 1992.

[10] David S. Taubman and Michael W. Marcellin. JPEG2000: standard for interactive imaging. *Proceedings of the IEEE*, 90(8):1336–1357, Aug 2002.

[11] Jared Stookey, Zhongyi Xie, Barbara Cutler, W. Randolph Franklin, Dan Tracy, and Marcus V. A. Andrade. Parallel ODETLAP for terrain compression and reconstruction. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, GIS '08, pages 17:1–17:9, New York, NY, USA, November 2008. ACM.

[12] Jared Stookey. Parallel terrain compression and reconstruction. Master's thesis, Rensselaer Polytechnic Institute, 2008.

[13] You Li and W. Randolph Franklin. 5D-ODETLAP: A novel high-dimensional compression method on time-varying geospatial data. In *Proceedings of AutoCarto 2010*, November 2010.

[14] You Li, Tsz-Yam Lau, Christopher S. Stuetzle, Peter Fox, and W. Randolph Franklin. 3D oceanographic data compression using 3D-ODETLAP. *SIGSPATIAL Special*, 2(3):7–12, November 2010.

[15] You Li. *CUDA-accelerated HD-ODETLAP: a high dimensional geospatial data compression framework*. PhD thesis, Rensselaer Polytechnic Institute, 2011.

[16] Amir Said and William A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, Jun 1996.

[17] Beong-Jo Kim and William A. Pearlman. An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In *Proceedings of the 1997 Data Compression Conference*, DCC '97, pages 251–260, March 1997.

[18] Daniel N. Benedetti, W. Randolph Franklin, and Wenli Li. CUDA-accelerated ODETLAP: A parallel lossy compression implementation. In *23rd Fall Workshop on Computational Geometry*, FWCG '13, October 2013.

[19] Daniel N. Benedetti. CUDA-accelerated ODETLAP: a parallel lossy compression implementation for multidimensional data. Master's thesis, Rensselaer Polytechnic Institute, 2014.

[20] David Luebke, Mark Harris, Jens Krüger, Tim Purcell, Naga Govindaraju, Ian Buck, Cliff Woolley, and Aaron Lefohn. GPGPU: General purpose computation on graphics hardware. In *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, New York, NY, USA, August 2004. ACM.

[21] NVIDIA Corporation. CUDA parallel computing platform. http://www.nvidia.com/object/cuda_home_new.html, 2016. Accessed 2016-06-01.

[22] NVIDIA Corporation. GPU-accelerated libraries. https://developer.nvidia.com/gpu-accelerated-libraries, 2016. Accessed 2016-06-01.

[23] Nathan Bell and Jared Hoberock. Thrust: A productivity-oriented library for CUDA. *GPU Computing Gems Jade Edition*, pages 359–372, 2011.

[24] Steven Dalton, Nathan Bell, Luke Olson, and Michael Garland. Cusp: Generic parallel algorithms for sparse matrix and graph computations. http://cusplibrary.github.io/, 2015. Accessed 2016-06-01.

[25] Jonathan R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[26] AIRS Science Team/Joao Texeira (2013). Aqua AIRS Level 3 Monthly Standard Physical Retrieval (AIRS+AMSU), version 006, Greenbelt, MD, USA, NASA Goddard Earth Science Data and Information Services Center (GES DISC). doi:10.5067/AQUA/AIRS/DATA319. Accessed 2016-06-01.

[27] Daniel S. Marcus, Tracy H. Wang, Jamie Parker, John G. Csernansky, John C. Morris, and Randy L. Buckner. Open access series of imaging studies (OASIS): Cross-sectional MRI data in young, middle aged, nondemented, and demented older adults. *Journal of Cognitive Neuroscience*, 19(9):1498–1507, September 2007.

[28] Peter Schelkens, Adrian Munteanu, Alexis Tzannes, and Chris Brislawn. JPEG2000 Part 10 – volumetric data encoding. In *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems*, ISCAS '06, pages 3874–3877, May 2006.