In this Supplementary Material of appendices, all figure and equation numbers are continued from the paper.

Appendix A: Formal Proof of Theorem 2

Theorem 2. *Let $\tilde{F}$ be the piece-wise linear function returned by Algorithm 1 (represented by $S_{\tilde{F}}$). For any k, let F^* be the optimal interpolant with k pieces, defined in Problem 2. Then $\tilde{F}$ has* $\text{Cost}(\tilde{F}) \leq \text{Cost}(F^*) + Ek$ *and at most* $q = k + \frac{2\cdot\text{Cost}(F^*)}{E}$ *segments.*

Formal Proof of Theorem 2. In any piece-wise linear interpolation, each segment is determined by its start (time) point, so we will sometimes reference segments by their start points.

Let $1 = s_1^* \leq \ldots \leq s_k^*$ be the start points of the segments in F^* and let $(m_1^*, b_1^*), \ldots, (m_k^*, b_k^*)$ be the corresponding interpolation coefficients. Similarly, let $1 = \tilde{s}_1 \leq \ldots \leq \tilde{s}_q$ be the start points of the segments in $\tilde{F}$ and let $(\tilde{m}_1, \tilde{b}_1), \ldots, (\tilde{m}_q, \tilde{b}_q)$ be the corresponding interpolation coefficients. Note that these coefficient are chosen optimally for each segment (in Line 4 of Algorithm 1).

We split the segments of $\tilde{F}$ into two groups, G and H. As discussed before, let G contain any segment that is entirely "contained" within one of the segments of F^*, possibly sharing a start time point. Namely, the start point $\tilde{s}_i$ of a G segment must satisfy $s_j^* \leq \tilde{s}_i$ and $\tilde{s}_{i+1} - 1 < s_{j+1}^* - 1$ for some $j \in \{1, \ldots, k\}$. Recall from Eq. (4) that the segment of F^* starting at s_j^* ends at $s_{j+1}^* - 1$, and similarly the segment of $\tilde{F}$ starting at $\tilde{s}_i$ ends at $\tilde{s}_{i+1} - 1$. Let H contains all other segments – specifically, those that "cross" between segments in F^*, or share an end time point with a segment in F^*. The sets G and H are visualized in Figure 2.

With these definitions in place, we first bound $\text{Cost}(\tilde{F})$. We have:

$$\text{Cost}(\tilde{F}) = \sum_{\tilde{s}_i \in G} \text{Cost}_i(\tilde{F}) + \sum_{\tilde{s}_i \in H} \text{Cost}_i(\tilde{F}), \tag{9}$$

where $\text{Cost}_i(\tilde{F})$ is defined as $\text{Cost}_i(\tilde{F}) = \sum_{j=\tilde{s}_i}^{\tilde{s}_{i+1}-1} \left\|\tilde{m}_i \cdot j + \tilde{b}_i - Y_{j:}\right\|_2^2$. For the first term, we clearly have

$$\left[\sum_{\tilde{s}_i \in G} \text{Cost}_i(\tilde{F})\right] \leq \text{Cost}(F^*). \tag{10}$$

Specifically, consider any segment of F^*, s_j^*, with interpolation coefficients (m_j^*, b_j^*) and consider the G segments $\tilde{s}_w, \ldots, \tilde{s}_{w+z}$ contained in s_j^*. We have that:

$$\begin{aligned}
\sum_{i=w}^{w+z} \text{Cost}_i(\tilde{F}) &= \sum_{i=w}^{w+z} \left[\min_{m,b \in \mathbb{R}^N} \sum_{\ell=\tilde{s}_i}^{\tilde{s}_{i+1}-1} \|m \cdot \ell + b - Y_{\ell:}\|_2^2\right] \\
&\leq \sum_{i=w}^{w+z} \left[\sum_{\ell=\tilde{s}_i}^{\tilde{s}_{i+1}-1} \|m_j^* \cdot \ell + b_j^* - Y_{\ell:}\|_2^2\right] \\
&= \sum_{\ell=\tilde{s}_w}^{\tilde{s}_{w+z+1}-1} \|m_j^* \cdot \ell + b_j^* - Y_{\ell:}\|_2^2 \\
&\leq \sum_{\ell=s_j^*}^{s_{j+1}^*-1} \|m_j^* \cdot \ell + b_j^* - Y_{\ell:}\|_2^2 = \text{Cost}_j(F^*).
\end{aligned}$$

Summing over all j proves (10). Next, consider the second term of (9). There are k segments in H, and each must have cost $< E$, since otherwise it would have been terminated earlier by Algorithm 1. So we conclude that $\sum_{\tilde{s}_i \in H} \text{Cost}_i(\tilde{F}) < Ek$. Combined with (10), this yields the approximation guarantee of Theorem 2.

It remains to prove the upper bound on the number of segments q in $\tilde{F}$. This requires bounding the number of H segments (equal to k) and the number of G segments, which is more challenging. To do so, we continue with the notation above: let s_j^* be a segment of F^* and consider the G segments $\tilde{s}_w, \ldots, \tilde{s}_{w+z}$ contained in s_j^*. Let m_j^*, b_j^* be the interpolation coefficients for segment s_j^*. We claim that for all $i \in \{w, \ldots, w+z\}$:

$$\sum_{\ell=\tilde{s}_i}^{\tilde{s}_{i+1}} \left\|m_j^* \cdot \ell + b_j^* - Y_{\ell:}\right\|_2^2 \geq E. \tag{11}$$

This must be true because, otherwise, m_j^* and b_j^* provide a piecewise interpolation for all $\ell \in [\tilde{s}_i, \tilde{s}_{i+1}]$ with error $< E$, and thus we would have extended segment $\tilde{s}_i$ by at least one more data point when running Algorithm 1. I.e., we would have included $\tilde{s}_{i+1}$ in that segment..

Let $|G|$ denote the number of elements in the set of G segments. For any $\tilde{s}_i \in G$, let $M(i) = j$ if $\tilde{s}_i$ is contained in segment s_j^*. Summing (11) over all $j = 1, \ldots, k$ we have:

$$\sum_{\tilde{s}_i \in G} \left[\sum_{\ell=\tilde{s}_i}^{\tilde{s}_{i+1}} \left\|m_{M(i)}^* \cdot \ell + b_{M(i)}^* - Y_{\ell:}\right\|_2^2\right] \geq \sum_{\tilde{s}_i \in G} E = |G| \cdot E \tag{12}$$

At the same time, we have that

$$\begin{aligned}
&\sum_{\tilde{s}_i \in G} \left[\sum_{\ell=\tilde{s}_i}^{\tilde{s}_{i+1}} \left\|m_{M(i)}^* \cdot \ell + b_{M(i)}^* - Y_{\ell:}\right\|_2^2\right] \\
&\leq 2 \sum_{\tilde{s}_i \in G} \left[\sum_{\ell=\tilde{s}_i}^{\tilde{s}_{i+1}-1} \left\|m_{M(i)}^* \cdot \ell + b_{M(i)}^* - Y_{\ell:}\right\|_2^2\right] \leq 2 \cdot \text{Cost}(F^*)
\end{aligned} \tag{13}$$

The first inequality comes from the fact that, for each ℓ, $\left\|m_{M(i)}^* \cdot \ell + b_{M(i)}^* - Y_{\ell:}\right\|_2^2$ appears in the first equation *at most* twice: specifically, only if $\ell = \tilde{s}_i$ for some i. Otherwise it appears once. The second inequality holds because the sum is exactly the cost of F^* restricted to some subset of time points in our dataset – the total cost of F^* can only be higher. Combining (13) and (12), we have:

$$2 \cdot \text{Cost}(F^*) \geq |G| \cdot E$$

and thus $|G| \leq \frac{2\cdot\text{Cost}(F^*)}{E}$. Adding in the k segments in H, we conclude that the total number of segments produced by Algorithm 1 is $\leq k + \frac{2\cdot\text{Cost}(F^*)}{E}$. □

Appendix B: Final Greedy Algorithm

Below we provide formal pseudocode for the final greedy method with automatic parameter tuning, which is outlined in Section 3.4.2. Note that we take as convention that $\max(\{\}) = -\infty$ – i.e., the maximum value of the empty set should evaluate to $-\infty$. We also use an extending function signature for subroutine calls to Algorithm 1, BasicGreedy. In addition to passing in the time steps $t = [1, \ldots, T]$, the data matrix Y, and a threshold parameter E, we also assume the algorithm is modified to take a 4th parameter J (an integer in $\{1, \ldots, T\}$) and a fifth, Z, which itself is a piece-wise linear function (represented as a set of tuples). These parameters indicate that BasicGreedy should be initialized with starting solution Z and should only start its main loop (line 3 in Algorithm 1) at index $j = J$ — i.e., it should operate exactly as if for the first $J-1$ time steps the algorithm already produced solution Z, but has not yet "closed" the last segment in Z. Finally, to keep notation concise, in the pseudocode below, we assume that k piece-wise linear functions are specified simply by a list of the starting points of each segment. That representation is what is passed to the BasicGreedy algorithm with extended input signature. In reality, we would also need to pass in the optimal coefficients for each segment, which have already been computed by each call to the BestLinearFit() function.

Algorithm 2 Final Greedy Algorithm

Input: Time steps $[1,\dots,T] = t$, data matrix $Y \in \mathbb{R}^{T\times N}$, resolution $\sigma > 1$

Initialization: $\Lambda = \{\}$, $s \leftarrow 1$, ZeroErrorSolution $\leftarrow \{1\}$, $\text{Err}_{min} = \infty$

```
 1: for $j = 2,\dots,T$ do
 2:     $\text{Err}_{max} \leftarrow \text{BestLinearFit}\left([1,\dots,j],[Y_{1:};\dots;Y_{j:}]\right)$
 3:     if $\text{Err}_{max} = 0$ then
 4:         continue
 5:     end if
 6:     $\text{Err}_{local} \leftarrow \text{BestLinearFit}\left([s,\dots,j],[Y_{s:};\dots;Y_{j:}]\right)$
 7:     if $\text{Err}_{local} > 0$ then
 8:         $\text{Err}_{min} \leftarrow \min(\text{Err}_{local}, \text{Err}_{min})$
 9:         $\text{ZeroErrorSolution} \leftarrow \text{ZeroErrorSolution} \cup \{j\}$
10:         $s \leftarrow j$
11:     end if
12:     $\Lambda_{new} = \{\lfloor\log_\sigma(\text{Err}_{min})\rfloor,\dots,\lceil\log_\sigma(\text{Err}_{max})\rceil\}$
13:     for $L \in \Lambda_{new} \setminus \Lambda$ do
14:         if $L > \max(\Lambda)$ then
15:             $\text{BasicGreedy}(t, Y, \sigma^L, j+1, \{1\})$
16:         else if $L > \log_\sigma(\text{Err}_{min})$ then
17:             $Z = \text{ZeroErrorSolution} \setminus \{j\}$
18:             $\text{BasicGreedy}(t, Y, \sigma^L, j+1, Z)$
19:         else
20:             $\text{BasicGreedy}(t, Y, \sigma^L, j+1, \text{ZeroErrorSolution})$
21:         end if
22:     end for
23:     $\Lambda \leftarrow \Lambda_{new}$
24: end for
```

Efficient Implementation

Naively we may let each thread perform the tasks of **Basic Greedy**, including reading input from disk. However, this would cause I/O contention and thus a major slowdown if the threads share a single disk. Even on parallel disks, to achieve parallel disk reads, we would typically need to partition the input into subgroups and copy/move these subgroups among the disks, which is nontrivial and has an additional I/O overhead.

To address this issue, we observe that all threads actually read the *same* time-step data from disk, which is not really necessary — the key idea is that we can just let *one* thread perform the data reading, and *share* the data read among the threads.

In our implementation, we have three types of threads: (1) one *reading* thread that reads the input from disk, (2) one *main* thread that computes E_{max} and E_{min} (and their corresponding solutions) and also creates new threads if needed, and (3) additional threads that run **Basic Greedy** with thresholds in the range (E_{min}, E_{max}) (exclusive). In each iteration, the reading thread reads one time step from disk, which is shared among all threads (a shared variable) when the reading is done. Then all threads of types (2) and (3) proceed with their own computation on the current time step. When all such threads are done, we move on to the next iteration for the next time step. In this way, the total I/O time for reading the input is essentially the same as that of **Basic Greedy**, which is optimal.

Analysis of Runtime and Memory Footprint

Now we analyze the running time of our implementation. Let τ be the time of **Basic Greedy** to compute on a single time step (i.e., *not* including the disk reading time). In each iteration, the main thread would take about 2τ time to finish its computation; within the same 2τ time, all existing threads of type (3) would also finish the computation. If the main thread does not create any new threads (of type (3)), then this iteration is done. Otherwise, the newly created threads would take an additional τ time to finish. Therefore, the computing time for each iteration is about 2τ to 3τ, and thus the total computing time of our implementation of **Final Greedy** is about 2-3 times of that of **Basic Greedy**.

Next we analyze the memory footprint of our implementation. We first consider **Basic Greedy**. From Sec. 3.2, computing the optimal error Err^* (see Eq. (5)) only depends on the memory size of $A'Y$, which is a $2\times N$ matrix. To compute the optimal solution $Z^* = (A'A)^{-1}A'Y$ (see Eq. (3)) in a streaming way, we need to update $A'Y$ by multiplying the new column of A' with the new row of Y (which is the new time step read, an N-tuple R), and adding the result to the original $A'Y$ (a $2\times N$ matrix). To do so, let $[a \;\; b]'$ be the new column of A', and P and Q be the first and second rows of the original $A'Y$. Then we can update P to $P+aR$ as follows: each time we multiply a with an element of R we immediately add the result to the corresponding element of P; we update Q to $Q+bR$ in the same way. Therefore, we can update $A'Y$ without extra working space, using $2N$ scalar values for $A'Y$ and N scalar values for the new time step, in a total of $3N$ scalar values. In order to achieve more accurate results in matrix operations, we use data type *double* for all the scalar values involved, so the memory footprint M of **Basic Greedy** is $M = 3N$ doubles. Let s be the size of one time step in the input (N scalar values). If the input data type is double, then s corresponds to N doubles and M is $3s$. If the input data type is float (as in our experiments), then $M = 6s$.

Now we analyze the memory footprint M' of **Final Greedy**. Let Th be the total number of threads. The reading thread (type (1)) uses memory size of N doubles for the current time step. The main thread (type (2)) computes E_{max} and E_{min} (and their corresponding solutions), each using the memory size of $A'Y$ ($2N$ doubles), with a total of $4N$ doubles. For each of the remaining $(Th-2)$ threads (type (3)), the memory size of $A'Y$ ($2N$ doubles) is used. The total memory footprint is thus $M' = N + 4N + (Th-2)\cdot 2N = 2N\cdot Th + N$ doubles. Recall that the memory footprint of **Basic Greedy** is $M = 3N$ doubles, and thus we have $M'/M = (2\cdot Th+1)/3$. To express M' in terms of s, we have $M' = 2s\cdot Th + s$ if the input data type is double, and $M' = 4s\cdot Th + 2s$ if the input data type is float.

Appendix C: Images Rendered from Reconstruction

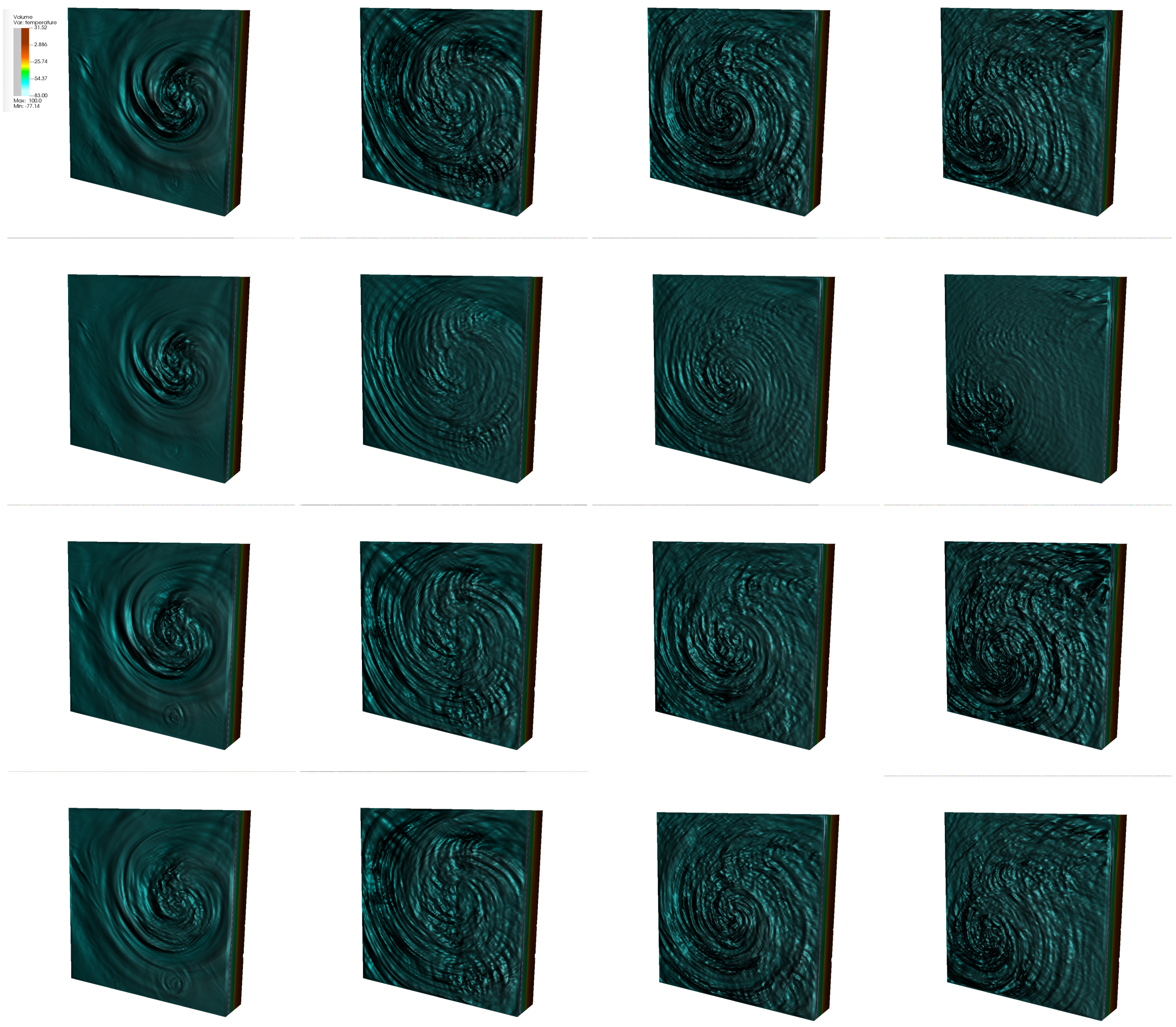

Figure 8: *Volume rendering of the reconstruction results of the* Isabel *dataset by* ***Final Greedy****,* ***AR-DP****, and* ***Sampling****, with the number of N-d data items stored* $= 14$ *(and the total relative errors (see Fig. 4(a)) 0.1456, 0.0888 and 0.2411 respectively), at time steps 6, 14, 28, 44 from left to right. Top row: original data; second row:* ***Final Greedy****; third row:* ***AR-DP****; bottom row:* ***Sampling****. The normalized root-mean-square error (NRMSE) values of the images from top down: left column (0, 4.86%, 8.72%, 8.73%), middle-left column (0, 9.74%, 12.47%, 12.61%), middle-right column (0, 9.75%, 12.09%, 12.43%), right column (0, 8.57%, 12.32%, 10.80%). NRMSE is defined as* $\frac{\sqrt{\sum_{i,j}\sum_{k=1}^{3}(A[i,j,k]-B[i,j,k])^2}}{\sqrt{\sum_{i,j}\sum_{k=1}^{3}A[i,j,k]^2}}$*, where A is the ground truth and B the image of reconstruction, and the R, G, B values of the* (i,j) *pixel of the image A are accessed by* $A[i,j,k]$ *for* $k = 1,2,3$.

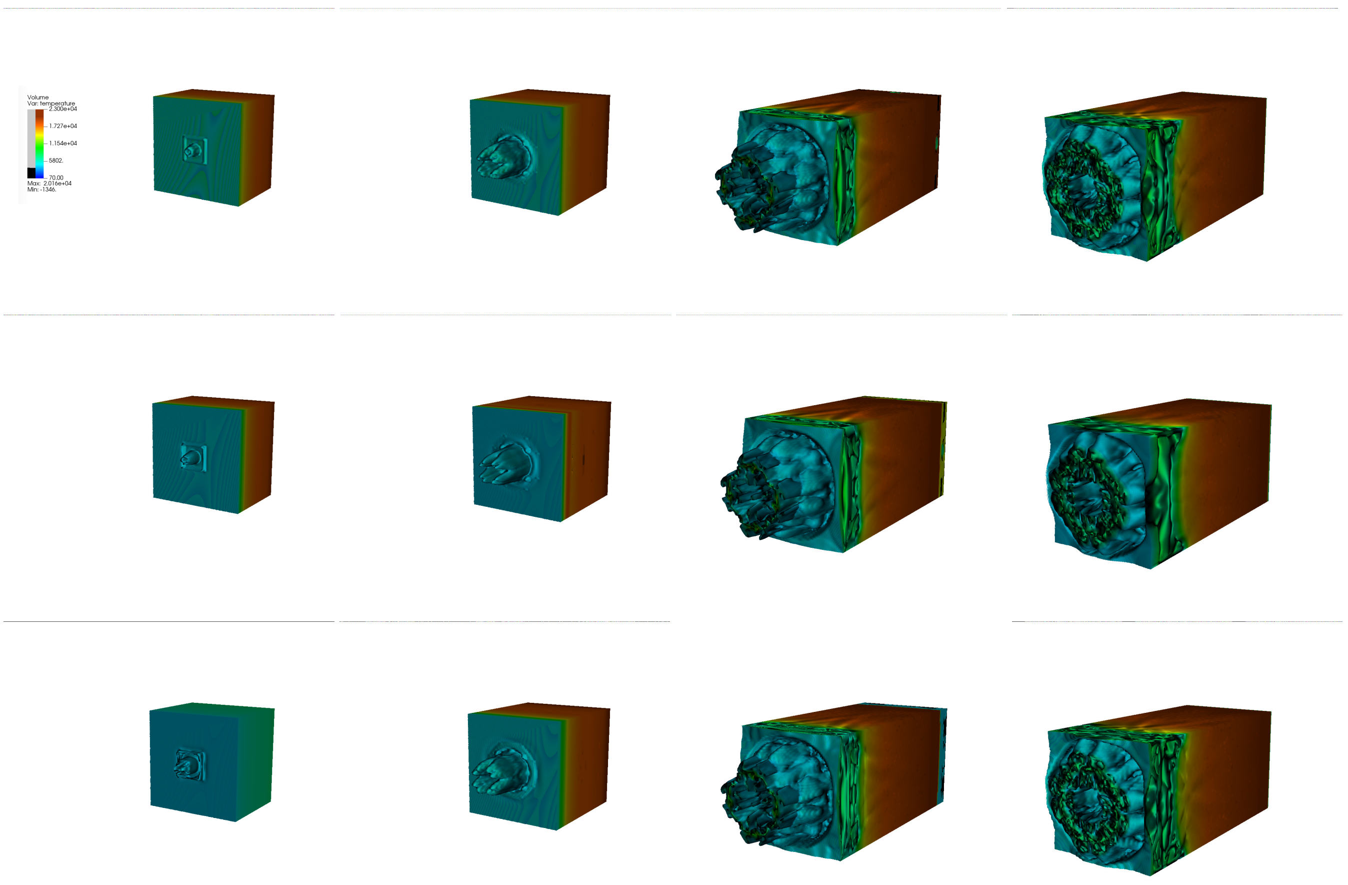

Figure 9: *Volume rendering of the reconstruction results of the* Radiation *dataset by* ***Final Greedy*** *and* ***Sampling****, with the number of N-d data items stored* = 24 *(and the total relative errors (see Fig. 5(a)) 0.0091 and 0.0235 respectively), at time steps 7, 48, 151, 186 from left to right. Top row: original data; middle row:* ***Final Greedy****; bottom row:* ***Sampling****. The NRMSE values of the images from top down: left column (0, 3.85%, 9.01%), middle-left column (0, 4.51%, 5.72%), middle-right column (0, 2.61%, 7.44%), right column (0, 7.57%, 7.96%).*