# Supplementary materials for “A Fast Exact Algorithm for Computing the Hypervolume Contributions in 4-D Space”

## S1. Updates for Inserting Once-dominated Gradients and Fully Covering Boxes in A Leaf Node

Section IV-D in the paper describes how to compute the changes of 2-D AGC if the weighted box to be inserted is a non-dominated gradient of a cell $C$. It is easy to be adopted for two other cases that may appear in our proposed HVC4D-G algorithm, i.e., the weighted box to be inserted is a once-dominated gradient or fully covers $C$.

In the case of the insertion of a once-dominated gradient, as shown in Fig. S1, the once-dominated gradient $H$ to be inserted is dominated by a single gradient $H_{1b}$. Therefore, we can entirely follow the method in Section IV-D to conduct updates in this case. Note that for this once-dominated case, only regions of type II in Fig. 9 in the paper needs to be constructed.

When the box $B$ to be inserted fully covers $C$, there are two possible cases, i.e., $B$ dominates some gradients and $B$ does not dominate any gradient. In this section we describe how to update when it dominates some gradients, whereas the other case is the case mentioned in the **Remark 2** in the paper (details of this case are presented in Section S2 in this supplementary materials). Figure S2 shows the update for vertical gradients in this case. It is seen that the division strategy in Section IV-D in the paper is still applicable and we obtain several regions of type II in Fig. 9 in the paper. After $B$ is inserted, all the gradients dominated by $B$ will be removed from data structures.

In summary, when the weighted box to be inserted is a once-dominated gradient or fully covers $C$, except for the leftmost and rightmost Region II's, all the subregions are constructed due to gradients in $G_1$ or $SubG_1$ dominated by the inserted box. Therefore, the number of subregions constructed remains $O(1)$ amortized and the time complexity remains $O(\log n)$ amortized as explained in Section S3 in this supplementary materials.

Besides, in the fully covering case, the $Pro(H_{1a})$ value of the lightest non-dominated gradient $H_{1a}$, i.e., 2-D HVC of the projection of $H_{1a}$ in the $x, w$-plane among vertical gradients regardless of horizontal gradients, needs to be updated. The update corresponds the region $D$ enclosed by green dashed lines in Fig. S1. $D$ is dominated by the projection of $H_{1a}$ in $x, w$-plane. Meanwhile, $D$ is also overlapped by the projection of $H_{1b}$ and a once-dominated gradient in $SubG_1$, so the volume of $D$ is not added into the $Pro(H_{1a})$ value before $B$ is inserted. After $B$ is inserted, $H_{1b}$ and the once-dominated gradient in $SubG_1$ are dominated by $B$ and will be removed. Therefore, the volume of $D$ should be added back for $H_{1a}$. The change of $Pro(H_{1a})$ leads to updating $AGC(H_{1a})$ and $Dom(H_{1a})$ as explained in **Remark 1** in the paper. This update can be charged to the update for the removal of a gradient dominated by $B$ (e.g., $H_{1b}$), so it does not increase the computational complexity.

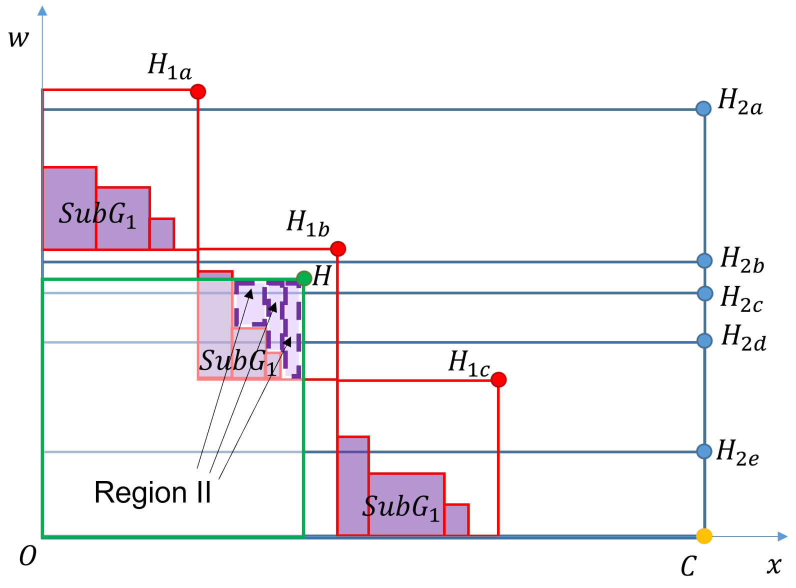


Fig. S1. The update for inserting a once-dominated vertical gradient. All gradients are projected vertically to $x, w$-plane. The division strategy in Section IV-D in the paper is still applicable and there are only Region II's in this case.

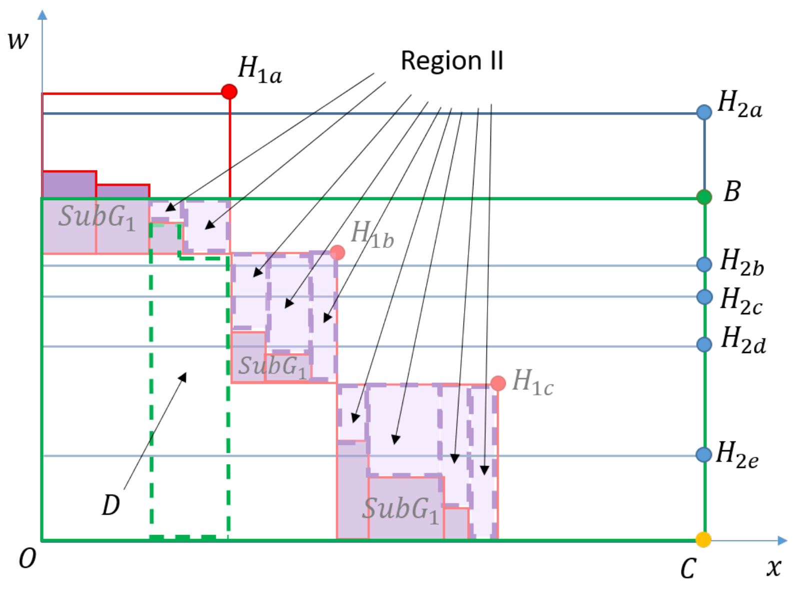


Fig. S2. The update for vertical gradients when a box $B$ that fully covers $C$ is inserted and $B$ dominates some gradients. All gradients are projected vertically to $x, w$-plane. The region $D$ enclosed by green dashed lines is the region dominated by $H_{1a}$ that is overlapped by vertical gradients dominated by $B$.

## S2. Updates For the Case in **Remarks 2**

In this section, we describe how to use *lazy* to maintain the intersection information for the case that a new NFB $\tilde{W}$ of a node $v$ is inserted such that $\tilde{w} < w_{\text{child}}$ and children of $v$ have no NFB/OFB, where $\tilde{w}$ is the weight of $\tilde{W}$. In this case, $\tilde{W}$ will intersect non-dominated gradients in leaf child nodes of $v$ (if any).

Fig. S3 shows an example of gradients in a child leaf node $s$ of $v$ (like Fig. 4(a) in the paper). In the figure, $C$ is the 2-D cell corresponding to $s$. The inserted $\tilde{W}$ fully covers $C$. Since $\tilde{w} < w_{\text{child}}$, $\tilde{W}$ only overlaps the contributions of the lightest non-dominated vertical and horizontal gradients, i.e., $H_{1c}$ and $H_{2c}$. Then, the overlapped contribution for the lightest non-dominated gradients is computed by Eq. (19) in the paper, where $[R(s) - A_2(s)]$ in Eq. (19) is the volume of region $D$ enclosed by dashed lines in Fig. S3. $D$ is the region covered by the lightest non-dominated gradients excluding those overlapped by other gradients. The volume of $D$ is trivial to compute. Since $D$ does not change after $\tilde{W}$ is inserted, it is sufficient to cumulate the change of *lazy* as in Eq. (20) in the paper at $v$ without visiting $s$.

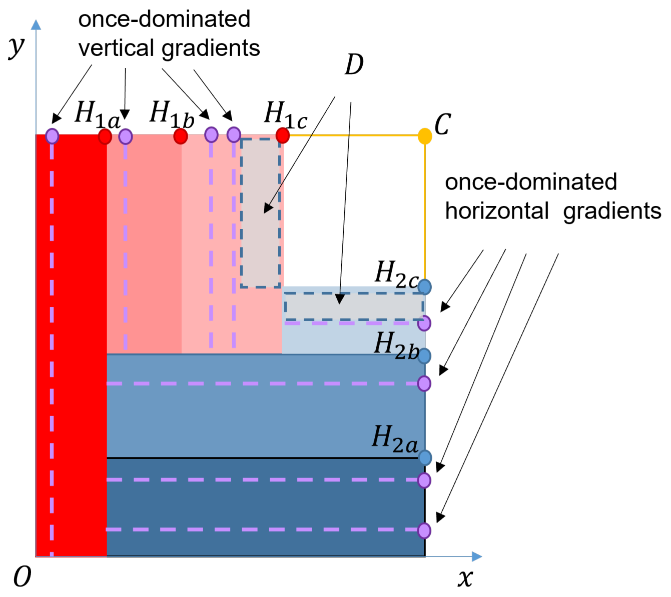

once-dominated vertical gradients
y
$D$
$H_{1a}$
$H_{1b}$
$H_{1c}$
$C$
once-dominated horizontal gradients
$H_{2c}$
$H_{2b}$
$H_{2a}$
$O$
$x$

Fig. S3. Illustration of the 2-D region $D$ in the $x, y$-plane that is covered by the lightest non-dominated gradient $H_{1c}$ and $H_{2c}$ excluding those covered by other gradients. Once-dominated gradients are marked by purple dashed lines.

## S3. Complexity Analysis and Theoretical Results For the Updates of 2-D AGCs

In Section IV-D, we described the update procedure for updating 2-D AGCs when a non-dominated vertical gradients is inserted. In this section, we present the complexity analysis for it and prove **Theorem 1** and **Corollary 1** in the paper.

For the updates of BSTs and BITs, similar to the YS algorithm, each vertical gradient is inserted and removed at most twice (for $G_1$ and $SubG_1$, respectively) during the whole procedure of the sweeping approach. Therefore, the number of updates for these data structures for each gradient is $O(1)$ amortized. This leads to an $O(\log n)$ amortized complexity.

For the complexity of computing changes of AGCs, we first show that the number of subregions constructed in Fig. 9 is at most $O(1)$ amortized. Except for the leftmost and rightmost subregions, each subregion is constructed due to a gradient in $G_1$ or $SubG_1$ dominated by $H$. After $H$ is inserted, the dominated gradient will be removed from $G_1$ (or $SubG_1$) and never be inserted back into $G_1$ (or $SubG_1$) again. Therefore, the number of subregions constructed for each dominated gradient is at most 4 (Regions I and II for $G_1$ and $SubG_1$) during the whole procedure of the sweeping approach. Thus, the number of constructed subregions is at most $O(1)$ amortized.

Then, we show that the updates in each subregion are of $O(\log n)$ amortized time with the help of BSTs and BITs. Each Region I involves one CIP for $H$, one range update of $Dom$ for horizontal gradients whose 2-D HVCs are fully covered by $H$ after projected onto the $y, w$-plane, and at most 2 CIPs for horizontal gradients whose 2-D HVCs are partially covered by $H$ after projected onto the $y, w$-plane. Each Region II involves one CIP for a gradient in $G_1$ and one single query and update of $Dom$. Therefore, all of these updates are of $O(\log n)$ amortized time.

As a result, the time complexity of the update procedure for each insertion is of $O(1) \cdot O(\log n) = O(\log n)$ amortized. This leads to the following result:

*Theorem 1:* **The changes of weighted contributions of a set of $n$ 2-D anchored gradients in a 2-D cell can be maintained in $O(\log n)$ amortized time per insertion, with a space complexity of $O(n)$.**

To apply the above result in HVC4D-G, note that $AGC(H')$ in Eq. (9) maintains 2-D AGCs, so its change corresponds to $\Delta_{\tilde{W}^j}$ in Eq. (6). There is only a factor difference $W_3^i$ between $\Delta_{\tilde{W}^j}$ and $\Delta_{\tilde{W}^j} W_3^i$ in Eq. (6), so we can replace the changes of $Con(H')$ in Eqs. (10)(12) and $\delta_{H'}$ in Eq. (8) with their changes multiplied by $W_3^i$ during the update. The 3-D weighted contribution in the leaf node can then be maintained. After the sweeping approach is finished, each 3-D weighted contribution is given by the revised $AGC(H')$ value with Eq. (9). This final computation costs extra $O(n \log n)$ time because the query of $Dom$ for each gradient has a time complexity of $O(\log n)$. It does not increase the whole time complexity as it is executed only once. Therefore, we have the following corollary:

*Corollary 1:* **For a set of $n$ 3-D weighted boxes whose first two dimensions are gradients of a 2-D cell:**

- **their weighted contributions in the cell can be maintained in $O(\log n)$ amortized time per insertion, with a data structure in $O(n)$ space;**
- **their weighted contributions in the cell can be computed in $O(n \log n)$ time and $O(n)$ space.**

## S4. Complexity Analysis and Theoretical Results For HVC4D-G

We can draw a similar complexity analysis for HVC4D-G (cf. Section III in the paper) as those for YS in [1] and prove **Theorem 2** in the paper.

The pre-processes in step (1) of HVC4D-G cost $O(n)$ time and the sorting in step 3(a) costs $O(n \log n)$ time. Building and initializing the binary space partition tree (BSP) in step (2) takes $O(n^{\frac{3}{2}} \log n)$ time and $O(n^{\frac{3}{2}})$ space [2].

For each iteration in step (4) of HVC4D-G, we will show that the four storage strategies of a BSP introduced in Section II-C are still valid in HVC4D-G, and thus each 2-D weighed box $\tilde{W}$ to be inserted is stored in at most $O(\sqrt{n} \log n)$ nodes. Consider the following relationships between $\tilde{W}$ and a node $v$ of the BSP:

1) $\tilde{W}$ does not intersect $v$. Its weighted contribution in $v$ is zero and it does not overlap weighted contributions of other boxes in $v$, so it is not stored in $v$ nor in children of $v$;
2) $\tilde{W}$ fully covers $v$. Its weighted contribution is given by Eq. (13) if it is an NFB of $v$. If it is an OFB of $v$, the weighted volumes overlapped by $\tilde{W}$ is computed based on Eq. (13) or (14) depending on whether $v$ has an NFB or not. Thus, $\tilde{W}$ is stored in $v$ but not in children of $v$;
3) $\tilde{W}$ partially covers $v$ and $v$ is not a leaf node. Its weighted contribution in $v$ is computed in children of $v$. The weighted volumes overlapped by $\tilde{W}$ are recursively obtained by Eq. (14), so $\tilde{W}$ is not stored in $v$;
4) $\tilde{W}$ partially covers $v$ and $v$ is a leaf node. In this case, $\tilde{W}$ defines a gradient of $v$ and it is stored in $v$.

Among nodes storing $\tilde{W}$, $\tilde{W}$ fully covers at most $O(\sqrt{n} \log n)$ nodes and partially covers $O(\sqrt{n})$ leaf nodes. For the fully covering case, the use of $w_{\text{child}}$ in Section V-B ensures that we have to traverse down the tree only when some boxes in children are dominated by $\tilde{W}$, so these traversals can be charged to the traversals for updating the dominated boxes. During the whole sweeping approach, the number of traversals for updating a box for a node itself cannot exceed 4 (once for insertion and once for removal respectively as a non-dominated box and as a once-dominated box). Since each box is stored in at most $O(\sqrt{n} \log n)$ nodes, the total number of traversals for updating one box is at most $4 \cdot O(\sqrt{n} \log n) = O(\sqrt{n} \log n)$. This is the same as that in YS, so both HVC4D-G and YS spend $O(\sqrt{n} \log n)$ amortized time in this case.

For the partially covering case, the insertion of a gradient in each leaf node costs $O(\log n)$ amortized time by the method proposed in Section IV, so the updates of this case are totally of $O(\sqrt{n}) \cdot O(\log n) = O(\sqrt{n} \log n)$ amortized time.

Besides, the maintenance of additional data in HVC4D-G includes the updates of $A_2, V_D, w_{\text{father}}$, and *lazy* values. They can be charged to the traversal for updating a box and the computational costs are of $O(1)$ time in each visited node given data of its adjacent father and children. Compared with YS, this only introduces different constant factors.

As a result, the update in step (4) for each box in HVC4D-G has the same asymptotic time complexity as that in YS for 4-D HV, which is in $O(\sqrt{n} \log n)$ amortized time.

The final traversal after the sweeping approach takes $O(1)$ time in each internal node and $O(\sqrt{n} \log n)$ time in each leaf node (obtaining $O(\sqrt{n})$ 3-D weighted contributions by Eq. (9)), so it costs $O(n^{\frac{3}{2}} \log n)$ time.

In summary, the construction, the updates, and the final traversal of the BSP are all in $O(n^{\frac{3}{2}} \log n)$ time. This leads to the main result of this paper as follows:

*Theorem 2:* **The 4-D ALLCONTRIBUTIONS problem can be computed in $O(n^{\frac{3}{2}} \log n)$ time and $O(n^{\frac{3}{2}})$ space.**

## S5. Complexity Analysis and Theoretical Results For HVC4D-GS

### A. Results

In Section VI-C, a new space partition strategy is proposed to improve the practical performance of HVC4D-G. Given a set of $n$ 2-D anchored boxes, the proposed strategy constructs a binary space partition tree (BSP). We will show the following two facts in Section S5-B:

1) The total number of piles in the leaf nodes of the BSP constructed by the proposed strategy is asymptotically no larger than that constructed by the original OY's space partition method, i.e., totally at most $O(n^{\frac{3}{2}})$ piles;
2) The height of the BSP is of $O(\log n)$, so the traversal overhead is $O(\log n)$ for each pile in the leaf nodes of the BSP constructed by the proposed strategy, asymptotically the same as that of the BSP constructed by OY's method.

The first fact ensures that the total number of gradients in the leaf nodes of the BSP constructed by HVC4D-G with the proposed strategy (HVC4D-GS) is at most $O(n^{\frac{3}{2}})$. Therefore, we can make the following analysis:

- **Complexity of building the BSP:** To store $O(n^{\frac{3}{2}})$ gradients in leaf nodes (i.e., the last level of the BSP), the time complexity and space complexity is $O(n^{\frac{3}{2}})$ and $O(n^{\frac{3}{2}})$, respectively. The height of the BSP is of $O(\log n)$, so the time complexity and space complexity of building the whole BSP is $O(n^{\frac{3}{2}} \log n)$ and $O(n^{\frac{3}{2}} \log n)$, respectively.
- **Complexity of updating the BSP during the sweeping approach:** In step (4) of HVC4D-G (as well as HVC4D-GS), weighted boxes are inserted into the BSP one by one in the sweeping approach. We discuss the total overhead of updating the BSP during the whole process of the sweeping approach in HVC4D-GS:[1]
  - In the case of the inserted weighted boxes partially covering leaf nodes, there are totally $O(n^{\frac{3}{2}})$ gradients to be inserted into leaf nodes during the whole procedure of the sweeping approach. To execute updates for these insertions, HVC4D-GS needs to **traverse the BSP** and **update contributions**. The former takes totally $O(n^{\frac{3}{2}} \log n)$ time because the overhead of each traversal is $O(\log n)$. The latter totally takes $O(n^{\frac{3}{2}} \log n)$ time as well because the costs of the updates are of $O(\log n)$ amortized time for each insertion of a gradient in a cell (**Corollary 1**).
  - In the case of the inserted weighted boxes fully covering arbitrary nodes, based on the four storage strategies introduced in Section II-C (as well as in Section S4), because there are totally $O(n^{\frac{3}{2}})$ gradients in leaf nodes and the height of the BSP is of $O(\log n)$, all the weighted boxes are stored totally in $O(n^{\frac{3}{2}} \log n)$ nodes of the BSP. They may be inserted and removed in each node at most twice during the whole process of the sweeping approach. If the inserted weighted boxes do not dominate gradients in leaf nodes, then the updates take $O(1)$ amortized time in each node. Otherwise, the updates for removing each dominated gradient take $O(\log n)$ amortized time (cf. Section S1). Since the dominated gradients will be permanently removed from the BSP and there are totally $O(n^{\frac{3}{2}})$ gradients, the total costs for removing dominated gradients are also of $O(n^{\frac{3}{2}} \log n)$ time.
- **Complexity of the final traversal of the BSP after the sweeping approach:** It takes $O(1)$ time in each internal node and $O(n^{\frac{3}{2}} \log n)$ time for all the gradients in all the leaf nodes (obtaining $O(n^{\frac{3}{2}})$ 3-D weighted contributions by Eq. (9)), so it costs $O(n^{\frac{3}{2}} \log n)$ time.

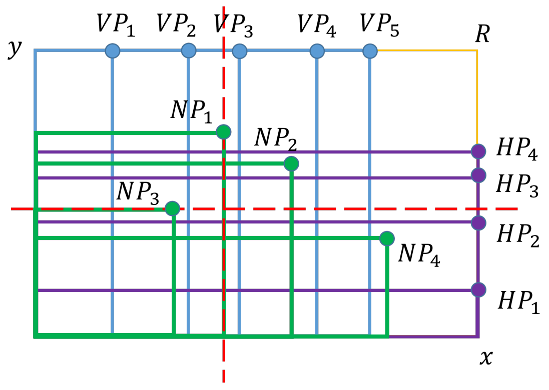
In summary, the construction, the updates, and the final traversal of the BSP are all in $O(n^{\frac{3}{2}} \log n)$ time. Therefore, the time complexity and space complexity of HVC4D-GS is $O(n^{\frac{3}{2}} \log n)$ and $O(n^{\frac{3}{2}} \log n)$, respectively.

The above result is almost the same as that of HVC4D-G in **Theorem 2**. Though the theoretical space complexity is a little worse, the experimental results suggest that HVC4D-GS often requires much less memory than HVC4D-G in practice (Table S6).

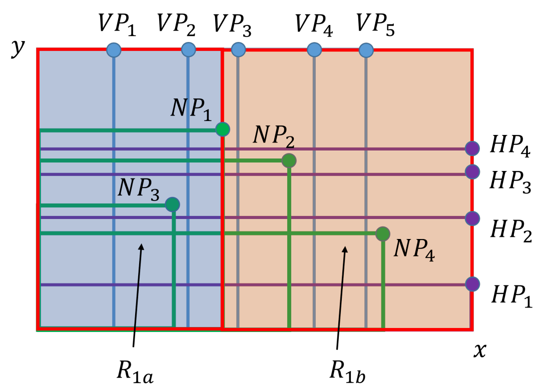
[1]It should be noted that our proposed strategy may not guarantee the second property of **Lemma 1**, so we have to discuss the overhead on a whole.

## *B. Analysis*

We consider a general case in the construction of the BSP as shown in Fig. S4. Suppose an arbitrary box $R \subset \mathbb{R}^2$, corresponding to an internal node in the BSP constructed by OY's method. There are $n_1$ vertical piles $VP = \{VP_1, VP_2, \ldots, VP_{n_1}\}$, $n_2$ horizontal piles $HP = \{HP_1, HP_2, \ldots, HP_{n_2}\}$, and $q$ ($1 \le q \le \sqrt{n}$) non-piles $NP = \{NP_1, NP_2, \ldots, NP_q\}$ partially covering $R$, where vertical piles fully cover $R$ in the second dimension ($y$-axis), horizontal piles fully cover $R$ in the first dimension ($x$-axis), and non-piles partially cover $R$ in both dimensions. In the example of Fig. S4(a), it is seen that $VP = \{VP_1, VP_2, \ldots, VP_5\}, HP = \{HP_1, HP_2, \ldots, HP_4\}, NP = \{NP_1, NP_2, \ldots, NP_4\}$.



(a) Region $R$



(b) Subregions of $R$ constructed by the new space partition strategy

Fig. S4. Illustration of (a) a region $R$ partially covered by vertical piles, horizontal piles and non-piles and (b) subregions $R_{1a}$ and $R_{1b}$ of $R$ constructed by the proposed space partition strategy. The proposed strategy chooses one splitting mechanism from two splitting mechanisms, i.e., splitting $R$ at $NP_1$ in the first dimension or splitting $R$ at $NP_3$ in the second dimension (red dashed lines in (a)), such that the number of piles and non-piles in subregions is smaller.

For this space partition problem, since $q \le \sqrt{n}$, HVC4D-GS replaces OY's method with the proposed space partition strategy and splits $R$ into cells. Without loss of generality, we assume that the upper bounds of boxes in $VP \cup HP \cup NP$ are distinct. In the following, the first/second upper bound of a box is denoted by an extra lower subscript, e.g., $(NP_1)_1$ means the first upper bound of $NP_1$.

We will prove the following three results:

1) The proposed strategy partitions $R$ into $q+1$ disjoint boxes $R = \cup_{i=1}^{q+1} C^{(i)}$ such that for each $C^{(i)}$, boxes in $VP \cup HP \cup NP$ either fully cover $C^{(i)}$ or become piles of $C^{(i)}$. Thus, each $C^{(i)}$ becomes a cell defined in **Lemma 1**, corresponding to a leaf node of the BSP;
2) The height of the tree constructed for $R$ (it is a subtree of the whole BSP) is of $O(\log q)$; and
3) The total number of piles in the $q+1$ cells is of $O(n_1 + n_2 + q^2 + \min\{n_1, n_2\}q)$.

**Assuming that the above three results are true, we can then obtain the two facts raised in Section S5-A**. Firstly, noting that $q \le \sqrt{n}$, OY's method splits the first interval of $R$ into $O(\lceil \frac{n_1}{\sqrt{n}} \rceil + 1)$ subintervals and splits the second interval of $R$ into $O(\lceil \frac{n_2}{\sqrt{n}} \rceil + q)$ subintervals, in which $O(q)$ subintervals result from the $q$ non-piles. We first consider the case of $n_1, n_2 > \sqrt{n}$. In this case, $O(\lceil \frac{n_1}{\sqrt{n}} \rceil + 1) = O(\frac{n_1}{\sqrt{n}}), O(\lceil \frac{n_2}{\sqrt{n}} \rceil) = O(\frac{n_2}{\sqrt{n}})$. Then, the space partition has totally $O(\frac{n_1}{\sqrt{n}}(\frac{n_2}{\sqrt{n}} + q))$ cells, each of which stores at most $O(\sqrt{n})$ piles. Therefore, there are totally $O(n_1(\frac{n_2}{\sqrt{n}} + q))$ piles in the tree constructed for $R$ by OY's method. Since

$$\begin{aligned}
&n_1(\frac{n_2}{\sqrt{n}} + q) = \frac{1}{2} 2n_1 q + \frac{1}{2} \frac{2n_1 n_2}{\sqrt{n}} \\
\ge &\frac{1}{2} \min\{n_1, n_2\} q + \frac{1}{2} \sqrt{n} q + \frac{1}{2} \frac{\sqrt{n} n_1}{\sqrt{n}} + \frac{1}{2} \frac{\sqrt{n} n_2}{\sqrt{n}} \quad (n_1, n_2 \ge \sqrt{n}) \\
\ge &\frac{1}{2} \min\{n_1, n_2\} q + \frac{1}{2} q^2 + \frac{1}{2}(n_1 + n_2) \quad (q \le \sqrt{n}) \\
= &\Omega(n_1 + n_2 + q^2 + \min\{n_1, n_2\} q),
\end{aligned} \tag{S-1}$$

we conclude that the total number of piles in the leaf nodes of the BSP constructed by the proposed strategy is asymptotically no more than that constructed by OY's method.

In the case of $n_1 > \sqrt{n} \ge n_2$ and $n_2 > \sqrt{n} \ge n_1$, the number of cells generated by OY's method is of $O(\frac{n_1}{\sqrt{n}} q)$ and $O(\frac{n_2}{\sqrt{n}} + q)$, respectively, and each cell stores $O(\sqrt{n})$ piles. In the case of $n_1, n_2 \le \sqrt{n}$, there will be $O(q)$ cells, and each cell stores $O(n_1 + n_2 + q)$ piles. Using similar techniques in Eq. (S-1) can obtain the same result as above.

Secondly, because the height of the tree constructed for $R$ is of $O(\log q) = O(\log n)$, the traversal overhead of the updates for each pile in a leaf node is $O(\log n)$, which is asymptotically the same as that in the BSP constructed by OY's method.

In the following, we will prove the above three results.

### 1) *Properties of Subregions Constructed by the Proposed Strategy*

The proposed strategy first finds the two non-piles $NP_i, NP_j \in NP$ such that $(NP_i)_1$ is the median value among all the first upper bounds of non-piles and $(NP_j)_2$ is the median value among all the second upper bounds of non-piles. It is possible that $i = j$. Based on $NP_i$ and $NP_j$, $VP, HP, NP$ are separated into several subsets:

$$\begin{aligned} VP^{(a)} &= \{P \in VP | P_1 < (NP_i)_1\}, VP^{(b)} = \{P \in VP | P_1 > (NP_i)_2\}, \\ HP^{(a)} &= \{P \in HP | P_2 < (NP_j)_2\}, HP^{(b)} = \{P \in HP | P_2 > (NP_j)_2\}, \\ NP^{(1a)} &= \{P \in NP | P_1 < (NP_i)_1\}, NP^{(1b)} = \{P \in NP | P_1 > (NP_i)_1\}, \\ NP^{(2a)} &= \{P \in NP | P_2 < (NP_j)_2\}, NP^{(2b)} = \{P \in NP | P_2 > (NP_j)_2\}. \end{aligned} \tag{S-2}$$

Let $v_a = |VP^{(a)}|, v_b = |VP^{(b)}|, h_a = |HP^{(a)}|, h_b = |HP^{(b)}|, q_{1a} = |NP^{(1a)}|, q_{1b} = |NP^{(1b)}|, q_{2a} = |NP^{(2a)}|, q_{2b} = |NP^{(2b)}|$. Then, we have

$$v_a + v_b = n_1, \quad h_a + h_b = n_2, \quad q_{1a} = q_{2a} = \lceil \frac{q}{2} \rceil - 1, \quad q_{1b} = q_{2b} = q - \lceil \frac{q}{2} \rceil.$$

The last two equality is because $(NP_i)_1$ and $(NP_j)_2$ are the median values. Note that $q_{1a} + q_{1b} = q_{2a} + q_{2b} = q - 1$ due to $NP_i \notin NP^{(1a)} \cup NP^{(1b)}$ and $NP_j \notin NP^{(2a)} \cup NP^{(2b)}$.

In the example of Fig. S4(a), $(NP_1)_1$ is the median value among all the first upper bounds of non-piles and $(NP_3)_2$ is the median value among all the second upper bounds of non-piles. According to Eq. (S-2), we have $VP^{(a)} = \{VP_1, VP_2\}$, $VP^{(b)} = \{VP_3, VP_4, VP_5\}$, $HP^{(a)} = \{HP_1, HP_2\}$, $HP^{(b)} = \{HP_3, HP_4\}$, $NP^{(1a)} = \{NP_3\}$, $NP^{(1b)} = \{NP_2, NP_4\}$, $NP^{(3a)} = \{NP_4\}$, $NP^{(3b)} = \{NP_1, NP_2\}$.

Then, the proposed strategy calculates the number of piles and non-piles in two subregions of $R$ respectively in two splitting mechanisms, i.e., splitting $R$ at $NP_i$ in the first dimension and splitting $R$ at $NP_j$ in the second dimension. The number of piles and non-piles in subregions of $R$ can be calculated as follows:

- If we split $R$ at $NP_i$ in the first dimension, $R$ is split into two subregions $R_{1a}$ and $R_{1b}$ (Fig. S4(b) shows the two subregions $R_{1a}$ and $R_{1b}$ after splitting $R$ at $NP_1$ in the first dimension). Then, piles in $VP^{(a)}$ become vertical piles of $R_{1a}$ and they do not intersect $R_{1b}$, piles in $VP^{(b)}$ become vertical piles of $R_{1b}$ and they fully cover $R_{1a}$, and piles in $HP$ become horizontal piles of both $R_{1a}$ and $R_{1b}$. Non-piles in $NP^{(1a)}$ become non-piles of $R_{1a}$ and they do not intersect $R_{1b}$, non-piles in $NP^{(1b)}$ become horizontal piles of $R_{1a}$ and non-piles of $R_{1b}$, while $NP_i$ becomes a horizontal pile of $R_{1a}$ and it does not intersect $R_{1b}$. As a result, $R_{1a}$ has $v_a$ vertical piles, $n_2 + q_{1b} + 1$ horizontal piles and $q_{1a}$ non-piles, while $R_{1b}$ has $v_b$ vertical piles, $n_2$ horizontal piles and $q_{1b}$ non-piles. **Totally** $n_1 + 2n_2 + q - \lceil \frac{q}{2} \rceil + 1$ **piles and** $q - 1$ **non-piles.**
- Analogously, if we split $R$ at $NP_j$ in the second dimension, leading to two subregions $R_{2a}, R_{2b}$ of $R$, then $R_{2a}$ has $n_1 + q_{2b} + 1$ vertical piles, $h_a$ horizontal piles and $q_{2a}$ non-piles, while $R_{2b}$ has $n_1$ vertical piles, $h_b$ horizontal piles and $q_{2b}$ non-piles. **Totally** $2n_1 + n_2 + q - \lceil \frac{q}{2} \rceil + 1$ **piles and** $q - 1$ **non-piles.**

In the example of Fig. S4(b), after $R$ is split at $NP_1$ in the first dimension, $\{VP_1, VP_2\}$ consists of vertical piles of $R_{1a}$, $\{NP_1, NP_2, NP_4, HP_1, HP_2, HP_3, HP_4\}$ consists of horizontal piles of $R_{1a}$, and $\{NP_3\}$ consists of the non-pile of $R_{1a}$; on the other hand, $\{VP_3, VP_4, VP_5\}$ consists of vertical piles of $R_{1b}$, $\{HP_1, HP_2, HP_3, HP_4\}$ consists of horizontal piles of $R_{1b}$, and $\{NP_2, NP_4\}$ consists of non-piles of $R_{1b}$.

Finally, the proposed strategy selects one of $NP_i$ and $NP_j$ such that the total number of piles and non-piles is smaller. So the proposed strategy generates two disjoint subregions such that **there are totally** $\min\{n_1, n_2\} + n_1 + n_2 + q - \lceil \frac{q}{2} \rceil + 1$ **piles and** $q - 1$ **non-piles**.

In the example of Fig. S4(b), the proposed strategy selects $NP_1$ and splits $R$ at $NP_1$ in the first dimension.

### 2) *The Number of Cells*

As analyzed above, after each splitting, the total number of non-piles in all the disjoint subregions is decreased by one, and the number of disjoint subregions is increased by one. As there are totally $q$ non-piles intersecting $R$, the proposed space partition strategy will generate $q+1$ nodes in the last level (i.e., leaf nodes) and stop the space partition. Each node in the last level has no non-pile, corresponding to a cell defined in **Lemma 1**. Therefore, the proposed strategy partitions $R$ into $q + 1$ disjoint cells.

### 3) *The Height of the Tree*

After $R$ is split into two subregions, the number of non-piles intersecting two subregions is at most $\lceil \frac{q}{2} \rceil - 1$ and $q - \lceil \frac{q}{2} \rceil$, respectively, i.e., no more than $\lfloor \frac{q}{2} \rfloor$. This property is also valid when splitting subregions of $R$. After no more than $\lceil \log q \rceil$ times of splitting along different branches of the tree, the number of non-piles in subregions will be decreased to zero, corresponding to cells. As a result, the height of the tree constructed for $R$ is of $O(\log q)$.

### 4) *The Number of Piles*

After $R$ is split into two subregions, the total number of piles in two subregions increases from $n_1 + n_2$ to $n_1 + n_2 + \min\{n_1, n_2\} + q - \lceil \frac{q}{2} \rceil + 1$ and the number of non-piles in each subregion decreases from $q$ to no more than $\lfloor \frac{q}{2} \rfloor$. Therefore, after $q$ times of splitting in the tree constructed for $R$, the total number of piles in all the disjoint cells of $R$ (corresponding to leaf nodes of the tree) is bounded by $O(n_1 + n_2 + q^2 + \min\{n_1, n_2\}q)$.

## S6. Detailed descriptions for YS algorithm

It is mentioned in Section II-D in the paper that YS algorithm [1] takes $O(n^{\frac{d-3}{2}} \log^2 n)$ amortized time for each box in the sweeping approach. Here we give a detailed description for it. Our proposed HVC4D-G also applies many ideas in YS algorithm.

According to **Lemma 1** in Section II-C in the paper, each $(d-2)$-D weighted box $B$ may partially cover at most $O(n^{\frac{d-3}{2}})$ cells of the binary space partition tree (BSP) constructed by the space partition method. Each cell corresponds to a leaf node in the BSP. Therefore, $B$ is stored in at most $O(n^{\frac{d-3}{2}} \log n)$ nodes of the BSP based on the four storage strategies explained in Section II-C in the paper. Among these nodes, there may be some nodes fully covered by $B$. The update for $B$ includes the update in leaf nodes where $B$ becomes a gradient of the cell and the update in nodes where the nodes are fully covered by $B$.

For the partially covering updates in leaf nodes, since the height of the BSP is at most $O(\log n)$, visiting each leaf node takes $O(\log n)$ time, totally $O(n^{\frac{d-3}{2}} \log n)$ time. For each leaf node, paper [1] considers a structure that all the boxes are gradients of a cell and develops a systemic method for updating weighted volume of gradients (cf. Section 3 and Section 4 in [1]). The complexity of this method is $O(\log^2 n)$. As a result, the update for all the leaf nodes takes $O(n^{\frac{d-3}{2}} \log^2 n)$.

For the fully covered nodes, [1] notices the following two facts:

- A box that fully covers father nodes must be lighter than boxes that fully cover children nodes. Otherwise, boxes that fully cover children nodes can be removed because the weighted volume covered by them have been covered by the box in father nodes. Note that the covered weighted volume only needs to be calculated once.
- Each box is at most inserted and removed once for each single node.

Due to the above facts, when a box $B$ to be inserted fully covers a node $v$, if it is lighter than all the boxes in children, the YS algorithm does not need to traverse children; otherwise, all the boxes in children lighter than $B$ need to be removed. In the latter case, the traversals for children can be treated as the updates for removing some boxes in them. These boxes will not be inserted into the BSP again, so during the whole process of the sweeping approach, the updates for them are executed at most twice (once for insertion and once for removal). As a result, the time complexity of traversals due to $B$ can be treated as the time complexity of removing boxes, which doubles the time complexity of insertions. The constant 2 can be hidden in the big O notation, so it does not increase the asymptotic complexity.

## S7. Absolute and Relative Errors of HVC Values in the Numerical Experiments

Because float number `double` in C++ can only store 16 significant digits, if the absolute errors are below 1e-15, we believe that they are small enough and the implementations are correct. For relative errors, they are computed by absolute errors divided by the real HVC values. When the HVC value of a single point is very small, the difference between absolute and relative errors can be very large. In this case, the magnitude of relative errors cannot reflect the correctness of algorithms. For example, as shown in Table S1, when $n$ increases from $1,000$ to $70,000$, the minimal HVC of a single point in spherical, cliff and hard instances decreases respectively from 8e-7 to 5e-10, from 5e-8 to 1e-12, and from 1e-8 to 2e-10. Since the absolute errors almost do not change, the relative errors increase accordingly as $n$ increases, but this has nothing to do with the correctness of algorithms.

Table S1 and S2 present the maximal absolute and relative error of every single point of HVC4D-G/HVC4D-GS using `double` compared with HVC4D. From the results, the absolute errors of HVC4D-G and HVC4D-GS are no greater than 1e-15 on spherical and hard instances. For cliff instances, absolute errors of HVC4D-G are still no greater than 1.1e-15. As explained above, we are convinced that our implementations are correct on these instances.

However, the maximal absolute error of HVC4D-GS is about 100 times larger than that of HVC4D-G on cliff instances when $n \geq 10,000$. This is because the space partition strategy in HVC4D-GS tends to generate some leaf nodes with many gradients for cliff instances. This requires more updates in a single leaf node and leads to more rounding errors than HVC4D-G.

To verify the above explanation, we change the commonly used float number `double` (8 bytes) by `long double` (16 bytes) in HVC4D-G and HVC4D-GS and test them. The results are presented in Table S3 and S4. When using `long double`, the absolute errors of HVC4D-G and HVC4D-GS are no greater than 1e-15 for all the instances. This verifies the correctness of them. Of course, using `long double` leads to longer running time for basic operations and requires more storage. For example, though HVC4D-G can compute $70,000$ cliff instance when it uses `double`, it runs out of memory(>8GB) for this instance when it uses `long double`.

## S8. Standard Deviation Values of Running Time

The standard deviation values are presented in Table S5 in this supplementary materials for running time data reported in Table II in the paper.

## S9. Comparison of Space Partition Tree Between HVC4D-G and HVC4D-GS

We record the number of gradients in the BSPs generated by HVC4D-G and HVC4D-GS on test instances with $30,000$ points. The results are presented in Table S6.

TABLE S1
THE MAXIMAL ABSOLUTE ERRORS OF SINGLE POINT BETWEEN HVC4D-G/HVC4D-GS USING `double` AND HVC4D ON SPHERICAL, CLIFF AND HARD INSTANCES. MIN HVC IS THE MINIMAL HVC OF A SINGLE POINT IN A POINT SET. OOM MEANS OUT OF MEMORY.

| Instance | Algorithm | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 | 20000 | 30000 | 50000 | 70000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Spherical | HVC4D-G | 5.990e-17 | 9.270e-17 | 8.080e-17 | 1.319e-16 | 1.257e-16 | 1.275e-16 | 1.502e-16 | 2.411e-16 | 2.083e-16 | 3.444e-16 |
| | HVC4D-GS | 6.798e-17 | 7.904e-17 | 1.128e-16 | 1.098e-16 | 8.182e-17 | 1.342e-16 | 2.298e-16 | 1.453e-16 | 1.598e-16 | 1.094e-16 |
| | Min HVC | 7.782e-07 | 1.726e-07 | 7.764e-08 | 4.192e-08 | 3.650e-08 | 8.573e-09 | 1.835e-09 | 2.330e-09 | 1.662e-10 | 4.678e-10 |
| Cliff | HVC4D-G | 1.432e-16 | 1.300e-16 | 2.247e-16 | 2.456e-16 | 3.330e-16 | 4.143e-16 | 2.847e-16 | 5.257e-16 | 1.093e-15 | 7.219e-16 |
| | HVC4D-GS | 2.668e-15 | 3.664e-15 | 3.786e-15 | 6.903e-15 | 8.523e-15 | 9.630e-15 | 2.074e-14 | 1.926e-14 | 1.321e-13 | 4.678e-14 |
| | Min HVC | 4.760e-08 | 6.960e-09 | 5.459e-09 | 6.454e-10 | 7.875e-10 | 1.281e-10 | 2.986e-11 | 1.486e-11 | 2.057e-12 | 1.362e-12 |
| Hard | HVC4D-G | 3.642e-17 | 4.395e-17 | 9.361e-17 | 5.877e-17 | 6.128e-17 | 9.337e-17 | 1.119e-16 | 1.416e-16 | 1.133e-16 | OOM |
| | HVC4D-GS | 3.624e-17 | 4.395e-17 | 9.354e-17 | 5.878e-17 | 1.120e-16 | 2.320e-16 | 9.908e-16 | 4.594e-16 | 6.271e-16 | 9.025e-16 |
| | Min HVC | 9.577e-07 | 2.391e-07 | 1.062e-07 | 3.822e-08 | 1.950e-08 | 9.553e-09 | 2.388e-09 | 1.061e-09 | 3.820e-10 | 1.949e-10 |

TABLE S2
THE MAXIMAL RELATIVE ERRORS OF SINGLE POINT BETWEEN HVC4D-G/HVC4D-GS USING `double` AND HVC4D ON SPHERICAL, CLIFF AND HARD INSTANCES. OOM MEANS OUT OF MEMORY.

| Instance | Algorithm | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 | 20000 | 30000 | 50000 | 70000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Spherical | HVC4D-G | 1.291e-12 | 1.796e-12 | 4.251e-12 | 2.562e-11 | 1.666e-11 | 2.646e-11 | 5.615e-11 | 9.068e-11 | 2.683e-10 | 2.832e-10 |
| | HVC4D-GS | 5.620e-12 | 3.495e-12 | 5.352e-12 | 1.746e-11 | 2.332e-11 | 3.513e-10 | 1.006e-10 | 1.501e-10 | 3.254e-10 | 4.610e-10 |
| Cliff | HVC4D-G | 3.477e-10 | 2.150e-09 | 4.963e-09 | 1.824e-08 | 6.632e-08 | 2.600e-07 | 1.265e-06 | 8.026e-07 | 1.164e-05 | 1.151e-05 |
| | HVC4D-GS | 1.528e-09 | 3.997e-09 | 9.184e-09 | 2.512e-08 | 1.777e-07 | 1.594e-07 | 2.176e-06 | 1.510e-06 | 8.084e-05 | 1.979e-05 |
| Hard | HVC4D-G | 3.034e-11 | 1.562e-10 | 7.174e-10 | 1.226e-09 | 2.529e-09 | 8.029e-09 | 3.804e-08 | 1.059e-07 | 2.367e-07 | OOM |
| | HVC4D-GS | 3.019e-11 | 1.562e-10 | 7.169e-10 | 1.227e-09 | 2.530e-09 | 8.444e-09 | 3.804e-08 | 1.060e-07 | 2.367e-07 | 3.670e-06 |

## S10. RESULTS ON POINT SETS WITH DOMINATED POINTS

Some algorithms for the HV and ALLCONTRIBUTIONS problems cannot directly deal with point sets with dominated points. In this section, we test our algorithms for this situation.

To generate point sets with dominated points, we uniformly sample points in $[0, 1]^4$. Five different point sets with $5,000$ points are generated and tested. Other experimental settings are the same as those in Sections VII-A and VII-B in the paper. The correct 4-D HVC for each point is computed by definition of HVC in Eq. (1) in the paper using the HV4D$^+$ algorithm in [3]. In the experiment, if the 4-D HVC of a dominated point is reported to be $0$ by an algorithm, its relative error for this point is recorded as $0$, otherwise, its relative error is recorded as $1$.

Table S7 presents the maximal absolute and relative errors between HVC values obtained by HVC4D/HVC4D-G/HVC4D-GS (using `double`) and the correct HVC values. From the table, HVC4D cannot deal with point sets with dominated points, while HVC4D-G and HVC4D-GS work well in this case. The reason is that BSPs constructed by our algorithms store both non-dominated and once-dominated boxes in the first $(d-2)$ dimensions, so the volumes of regions overlapped by dominated points can be correctly computed.

## REFERENCES

[1] H. Yıldız and S. Suri, "Computing Klee's measure of grounded boxes," *Algorithmica*, vol. 71, pp. 307–329, 2015.
[2] M. H. Overmars and C.-K. Yap, "New upper bounds in Klee's measure problem," *SIAM Journal on Computing*, vol. 20, no. 6, pp. 1034–1045, 1991.
[3] A. P. Guerreiro and C. M. Fonseca, "Computing and updating hypervolume contributions in up to four dimensions," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 449–463, 2018.

TABLE S3
The maximal absolute errors of single point between HVC4D-G/HVC4D-GS using `long double` and HVC4D on spherical, cliff and hard instances. OOM means out of memory.

| Instance | Algorithm | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 | 20000 | 30000 | 50000 | 70000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Spherical | HVC4D-G | 5.969e-17 | 9.281e-17 | 8.050e-17 | 1.319e-16 | 1.240e-16 | 1.267e-16 | 1.450e-16 | 2.416e-16 | 1.992e-16 | 3.442e-16 |
|  | HVC4D-GS | 6.885e-17 | 8.359e-17 | 1.127e-16 | 1.081e-16 | 7.568e-17 | 1.274e-16 | 2.332e-16 | 1.379e-16 | 1.597e-16 | 1.127e-16 |
| Cliff | HVC4D-G | 1.337e-16 | 1.001e-16 | 1.384e-16 | 1.175e-16 | 1.337e-16 | 1.335e-16 | 1.351e-16 | 1.462e-16 | 1.362e-16 | OOM |
|  | HVC4D-GS | 1.337e-16 | 1.001e-16 | 1.385e-16 | 1.175e-16 | 1.337e-16 | 1.335e-16 | 1.349e-16 | 1.462e-16 | 1.362e-16 | 1.322e-16 |
| Hard | HVC4D-G | 3.622e-17 | 4.396e-17 | 9.357e-17 | 5.878e-17 | 6.128e-17 | 9.335e-17 | 1.119e-16 | 1.416e-16 | 1.133e-16 | OOM |
|  | HVC4D-GS | 3.622e-17 | 4.396e-17 | 9.357e-17 | 5.878e-17 | 6.128e-17 | 9.335e-17 | 1.119e-16 | 1.416e-16 | 1.133e-16 | 9.025e-16 |

TABLE S4
The maximal relative errors of single point between HVC4D-G/HVC4D-GS using `long double` and HVC4D on spherical, cliff and hard instances. OOM means out of memory.

| Instance | Algorithm | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 | 20000 | 30000 | 50000 | 70000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Spherical | HVC4D-G | 4.127e-13 | 1.509e-12 | 2.613e-12 | 4.329e-12 | 5.833e-12 | 6.827e-12 | 1.476e-11 | 2.114e-11 | 5.161e-11 | 8.398e-11 |
|  | HVC4D-GS | 3.361e-13 | 9.439e-13 | 5.380e-12 | 2.908e-12 | 6.078e-12 | 1.549e-11 | 1.663e-11 | 4.360e-11 | 7.181e-11 | 4.628e-11 |
| Cliff | HVC4D-G | 1.521e-10 | 8.039e-10 | 1.885e-09 | 8.453e-09 | 1.120e-08 | 3.317e-08 | 2.482e-07 | 3.887e-07 | 1.303e-06 | OOM |
|  | HVC4D-GS | 1.521e-10 | 8.039e-10 | 1.885e-09 | 8.450e-09 | 1.120e-08 | 3.318e-08 | 2.482e-07 | 3.887e-07 | 1.299e-06 | 1.423e-06 |
| Hard | HVC4D-G | 3.017e-11 | 1.562e-10 | 7.171e-10 | 1.227e-09 | 2.529e-09 | 8.027e-09 | 3.804e-08 | 1.060e-07 | 2.367e-07 | OOM |
|  | HVC4D-GS | 3.018e-11 | 1.562e-10 | 7.171e-10 | 1.227e-09 | 2.529e-09 | 8.027e-09 | 3.804e-08 | 1.060e-07 | 2.367e-07 | 3.670e-06 |

TABLE S5
Standard deviation values of running time (in seconds) of HVC4D/HVC4D-G/HVC4D-GS using `double` for spherical, cliff and hard instances. OOM means out of memory.

| Instance | Algorithm | 1000 | 2000 | 3000 | 5000 | 7000 | 10000 | 20000 | 30000 | 50000 | 70000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Spherical | HVC4D | 0.0002 | 0.0005 | 0.0014 | 0.0024 | 0.0028 | 0.0043 | 0.0074 | 0.0137 | 0.0712 | 0.2045 |
|  | HVC4D-G | 0.0045 | 0.0051 | 0.0082 | 0.0189 | 0.0304 | 0.0137 | 0.0315 | 0.0226 | 0.0451 | 0.1017 |
|  | HVC4D-GS | 0.0031 | 0.0074 | 0.0078 | 0.0093 | 0.0079 | 0.0134 | 0.0257 | 0.0229 | 0.0503 | 0.1065 |
| Cliff | HVC4D | 0.0020 | 0.0038 | 0.0142 | 0.0643 | 0.1476 | 0.3277 | 1.3694 | 3.4043 | 19.8279 | 70.4817 |
|  | HVC4D-G | 0.0126 | 0.0723 | 0.1383 | 0.3073 | 0.5381 | 1.0347 | 4.3685 | 8.0736 | 18.1701 | 39.4242 |
|  | HVC4D-GS | 0.0140 | 0.0435 | 0.1049 | 0.2530 | 0.3683 | 0.6036 | 1.9942 | 4.0641 | 9.7234 | 17.6976 |
| Hard | HVC4D | 0.0015 | 0.0065 | 0.0112 | 0.0362 | 0.0776 | 0.1736 | 0.6962 | 1.5917 | 5.7891 | 19.8916 |
|  | HVC4D-G | 0.0076 | 0.0198 | 0.0386 | 0.0858 | 0.1516 | 0.2685 | 1.0660 | 2.3897 | 6.1265 | OOM |
|  | HVC4D-GS | 0.0035 | 0.0046 | 0.0048 | 0.0083 | 0.0148 | 0.0197 | 0.0412 | 0.0577 | 0.1485 | 0.1345 |

TABLE S6
Average number of gradients in the BSP constructed by HVC4D-G and HVC4D-GS on spherical, cliff and hard instances with $30,000$ points.

|  | Algorithm | Spherical | Cliff | Hard |
|---|---|---|---|---|
| Number of gradients | HVC4D-G | 1.11e+06 | 5.57e+06 | 5.57e+06 |
|  | HVC4D-GS | 7.75e+05 | 3.48e+06 | 5.10e+05 |

TABLE S7
The maximal absolute and relative errors of single point between HVC4D/HVC4D-G/HVC4D-GS using `double` and 4-D HVC values computed by definition on $5,000$ point sets with dominated points.

| Instance | No. of non-dominated points | Maximal absolute error |  |  | Maximal relative error |  |  |
|---|---|---|---|---|---|---|---|
|  |  | HVC4D | HVC4D-G | HVC4D-GS | HVC4D | HVC4D-G | HVC4D-GS |
| Set 1 | 163 | 8.597e-01 | 6.906e-15 | 6.847e-15 | 1.000e+00 | 1.895e-06 | 1.895e-06 |
| Set 2 | 165 | 1.022e+00 | 7.911e-15 | 7.909e-15 | 1.000e+00 | 3.159e-06 | 3.159e-06 |
| Set 3 | 108 | 5.928e+00 | 9.381e-15 | 9.383e-15 | 1.000e+00 | 1.191e-07 | 1.191e-07 |
| Set 4 | 142 | 9.510e-01 | 8.318e-15 | 8.293e-15 | 1.000e+00 | 4.762e-07 | 4.762e-07 |
| Set 5 | 124 | 1.773e+00 | 8.167e-15 | 8.161e-15 | 1.000e+00 | 2.256e-06 | 2.256e-06 |