n-Dimensional Sparse Voxel Trees for Compact Representation of Dimensionally Upscaled Voxelized Three-Dimensional Scenes

Branislav Madoš, Heidar Khorshidiyeh, Martin Chovanec, Norbert Ádám, Peter Poprík

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University in Košice, Letná 1/9, 042 00 Košice, Slovakia
branislav.mados@tuke.sk, heidar.khorshidiyeh@tuke.sk, martin.chovanec@tuke.sk , norbert.adam@tuke.sk, peter.poprik@tuke.sk

Abstract: This paper addresses the issue of how to represent the geometry of voxelized three-dimensional scenes using domain-specific hierarchical data structures. It provides a comprehensive overview of these data structures, delving into the specifics of sparse voxel octrees in greater depth. Then, it discusses the proposed use of dimensionality upscaling, i.e. the transformation of a three-dimensional voxelized scene into a higher number of dimensions, to enable the representation of the geometry of this scene via an n-dimensional sparse voxel tree hierarchical data structure. This approach allows a smaller size of binary representation of the 3D scene geometry – compared to its representation using a sparse voxel octree. This is documented in the next section of the paper, where we summarize the results of tests encoding the geometries of the respective scenes (based on voxelization of polygonal surface models), into sparse voxel trees of different levels of dimensionality. We achieved the highest compactness of the binary representation of the resulting data structure, compared to the traditional sparse voxel octree (increase in compactness ranged from 2.02 to 4.44 times), using upscaling to 6D and 7D, respectively.

Keywords: polygonal surface model; geometry representation of voxelized scene; lossless data compression; hierarchical data structure; sparse voxel octree

1 Introduction

When it comes to encoding of a binary representation of three-dimensional (3D) scenes, in the field of computer graphics, there is an approach that involves creating a 3D regular grid of volumetric picture elements (voxels). With this approach, individual voxels (vox) may be assigned attributes, e.g. color, transparency or material properties. To define the geometry of a voxelized 3D scene, one could use an approach where each voxel is assigned $1b$. This determines whether the corresponding voxel is passive (the bit is set to 0) or active (the bit is set to 1).

For 3D scenes with an $x \times y \times z$ voxel resolution, this approach produces a regular $x \times y \times z$ 3D grid of $1b$ scalar values. This represents a significant amount of data, as is clear from the example of a 3D scene with a resolution of $1024 \times 1024 \times 1024$ voxels ($1K^3$ vox), which requires a memory or secondary storage device space of $1Gb$ ($128MB$). When doubling the scene size (measured in number of voxels) in each of its axes ($2K^3$ vox), the data volume increases eight times. This underscores the need for more compact representation of this information through sophisticated ways of encoding, which are also forms of data compression.

Using universal compression algorithms for archiving or transmission (streaming) would allow a decrease in the amount of data required. However, this approach would not meet the expectation for an economic representation considering active use of these structures, when stored in limited storage - the computer's operating memory or the memory of the graphics card. A popular and frequently used solution that meets also this requirement is the use of domain-specific hierarchical data structures (HDSs) based on octant trees and directed acyclic graphs (DAGs), along with the corresponding encoding algorithms. These HDSs store information about voxels; the proportion of active voxels relative to their total number is expected to be low — frequently as low as 0.01% of voxels — indicating that the scene is sparse. Therefore, the corresponding HDSs are referred to as Sparse Voxel Octrees (SVOs), if they are based on octant trees, or Sparse Voxel Directed Acyclic Graphs (SVDAGs), if they are based on Directed Acyclic Graphs (DAGs).

Other HDSs have been derived from SVOs and SVDAGs, which have been further optimized through the implementation of additional features. These features help achieve increasingly compact encodings of the corresponding HDS at its binary level. These features include, for example, the application of a mirroring transformation or the introduction of multiple pointer types to child nodes with different binary representation lengths. This made possible the use of Frequency Based Compaction (FBC), which changes the order of nodes in the binary representation of the HDS, according to the frequency of their referencing. Details on this can be found in the Related Works section of this paper. The advantage of using these domain-specific HDSs is that they can be traversed quickly. The operations that need to be performed during it can be considered on-the-fly HDS decompression. Another advantage is the ability to traverse the HDS up to any selected level. This is useful for supporting the Level of Details (LOD) technique.

This paper is concerned with finding a more compact geometry of a voxelized 3D scene representation. This is accomplished by rearranging the voxels of scene into a regular grid with a higher dimensionality; then, a tree-based HDS with higher number of dimensions is created for the geometry representation purpose. The contribution of the paper is, therefore, in the proposal for the use of dimensionality upscaling of voxelized 3D scenes to higher number of dimensions to increase the compactness of their geometry representation through a representation using n-Dimensional Sparse Voxel Trees (nD SVTs) as a generalized form of Sparse Voxel Octrees.

2 Related Works

The use of hierarchical data structures as a means for economical representation of 2D data has been explored since the 1980s, when the use of quadtrees for image compression was addressed in [1, 2]. The use of the Common Subtree Merge (CSM) technique was also proposed [3]. The 2D Template-based Encoding (2DTE) data structure, which was introduced in [4] and extended to 3D in [5], was based on this principle. As with 2D images, the use of octant trees for the representation of 3D data was already explored in the 1980s [6-8].

Modern SVO-based HDSs implementing pointers to child nodes used to represent 3D scenes include Efficient Sparse Voxel Octrees (ESVOs), which were proposed by Laine and Karras in 2010 [9]. These have the advantage of having the capability to replace entire subtrees of a data structure with contour information, which is represented more economically at the binary level than the subtree being replaced. In 2022, Madoš et al. [10] introduced Clustered Sparse Voxel Octrees (CSVOs). This HDS offers a more significant compression of the 3D scene geometry by using three different node types. Each node type is optimized for both the number and the length of pointers to child nodes. In 2013, Baert et al. [11] introduced a two-step out-of-core algorithm that allows the voxelization of a mesh of triangles, thus obtaining an intermediate output, used to enable the efficient construction of SVO HDSs. In 2015, Pätzold and Kolb [12] presented a voxelization algorithm that produces SVOs directly, without an intermediate product.

DAG-based HDSs that use pointers to child nodes to represent 3D scenes include High Resolution Sparse Voxel Directed Acyclic Graphs, or HR SVDAGs, introduced by Kämpe et al. in 2013 [13]. These offer the possibility of compact representation of identical subtrees through the Common Subtree Merge (CSM) technique, where multiple instances of such a subtree are represented by only a single instance of the subtree that is multi-referenced. All subnodes and whole nodes of this HDS are aligned to 32b. In 2016, Villanueva et al. [14] introduced Symmetry-aware Sparse Voxel Directed Acyclic Graphs (SSVDAGs). This data structure enables CSM using a mirroring transformation and yields two pointer lengths to child nodes – 16b and 32b. This innovation allows for an optimization based on the frequency of referencing nodes – Frequency Based Compaction (FBC). HDSs based on DAGs, without any implementation of pointers to child nodes include Pointerless Sparse Voxel Directed Acyclic Graphs (PSVDAGs), introduced by Vokorokos et al. in 2020 [15]. Subsequently, an algorithm for fast transformation of PSVDAGs into SVDAGs was introduced by Madoš and Ádám in 2021 [16].

While the focus has been on HDSs with a lossless compression of 3D scene geometry and attributes in the past, there is also interest in lossy compression. In 2020, van der Laan et al. [17] introduced an HDS based on SVDAGs that modifies the subtrees to increase their degree of identity, thereby increasing the applicability of CSM. Thus, lossy data compression is applied, hence the name of the proposed HDS: Lossy Sparse Voxel Directed Acyclic Graphs (LSVDAGs).

The above HDSs are suitable for the representation of static scenes. However, even a small change in the scene geometry may require a significant change in the binary representation of the data structure, at the cost of its complete decompression, execution of the change and its subsequent recompression. To avoid this issue, Careil et al. [18] introduced an HDS called HashDAGs in 2020. This allows for interactive changes to the DAG data structure without decompression or recompression, making it compatible with the economical voxel attribute solution presented in [19].

Two approaches are used to extend the above-mentioned HDSs to represent additional attributes. The first introduces even more complex SVDAGs that integrate geometry and other attribute information into a single SVDAG. The second brings geometry compression using HDS with links to external data structures, carrying additional information about other attributes. Solutions integrating voxel color information, include MoxelDAG [20] and [18] [19] [21]. DAGs are also used to represent voxelized shadows in a compact form [22] [23].

3 Sparse Voxel Octrees

To efficiently encode SVO HDS, the dimensions of the represented 3D scene must be N^3; $N = 2^m$, where $m \geq 1$; $m \in \mathbb{N}$. The nodes forming the SVO are then divided hierarchically into m layers, which are assigned ordinal numbers from the range $< 0; m-1 >$. The layer of nodes containing the root node will have an ordinal number of 0, and the layer of nodes containing the leaf nodes (LNODEs) will have an ordinal number of $m-1$. Nodes stored in layers with an ordinal number from the interval $< 0; m-2 >$ are considered internal nodes (INODEs).

The 3D scene geometry is represented by a grid of N^3 values with $1b$ size. This grid will be represented in the adjacent SVO by a node embedded in a layer of nodes with ordinal number 0 (the root node). This node will contain an 8b Child Node Mask (CHNM) vector made up of eight 1b Header Tags (HTs). The CHNM is constructed by decomposing the grid associated with this node into eight child octants. For the root node, these child octants have $(\frac{N}{2})^3$ dimensions. For each child octant, it is determined whether it is passive or active. A passive octant has all its voxels passive (the corresponding geometry attribute is set to 0 for all its voxels). An active octant contains at least one active voxel, which has the corresponding geometry attribute set to 1. If the child octant is passive, it is assigned a 1b HT with value 0 in the SVO node being constructed. If the child octant is active, it is assigned a 1b HT with value 1 in the SVO node being constructed.

The order in which the child octants are ranked and the HTs are arranged in the CHNM can be a matter of choice. In this paper, we use the Morton order.

There is no further information provided for the passive child octant.

Further, a recursive decomposition procedure is applied to each active child octant, represented by the corresponding child grid, which origin was described above as the decomposition of its parent grid. The active child octant will therefore have a separate child node created and stored in a node layer of the data structure with an incremented ordinal number in comparison to the ordinal number of parent node layer (for the root node, this will result in child nodes stored in the node layer with ordinal number 1). Therefore, the recursive decomposition of the active child octants is executed, yielding nodes stored in layers with ordinal numbers from the interval $< 0; m - 1 >$.

The recursive decomposition stops at nodes stored in layer $m - 1$, which are called leaf nodes (LNODEs) and are related to the octants of $2 \times 2 \times 2$ voxels. The potential child nodes of leaf node, which should represent child octants formed by only a single voxel (a child octant of $1 \times 1 \times 1$ voxel), will not be created. The status of such voxel – whether it is passive (bit set to 0) or active (bit set to 1) – is recorded directly to the related HT in the CHNM of the LNODE.

The data structure must also represent the relationship between the corresponding HT set to 1 within the parent CHNM and its associated child node. This can be done without the use of pointers – leading to Pointerless Sparse Voxel Octrees (PSVOs), or with the use of pointers to child nodes – leading to Sparse Voxel Octrees (SVOs).

3.1 Pointerless Sparse Voxel Octrees

In the case of Pointerless Sparse Voxel Octrees, the relationship between the HT located in the CHNM of the parent node and the associated child node is expressed by the location of the binary representation of the child node immediately after the binary representation of the corresponding HT of the parent node, as shown in Figure 1. The size of a data structure's binary representation in bits is $8 \times p$, where p is the total number of nodes of the data structure. This is because each node includes only CHNM of 8b in size. While compact, this representation does not allow for a fast traversal of the tree.

To formalize the binary representation of PSVOs, we used the Backus-Naur Form:

PSVO ::= <NODE>

NODE ::= <INODE> | <LNODE>

INODE ::= (8)<HT>

HT ::= "0" | "1"<NODE>

LNODE ::= (8)<BIT>

BIT ::= "0" | "1"

where the following applies:

<SYM> - a mandatory non-terminal symbol SYM

"sym" - terminal symbol sym

(n)<SYM> - the SYM symbol, concatenated n-times

(n)*(m)<SYM> - the SYM symbol, concatenated n to m times

| - alternative

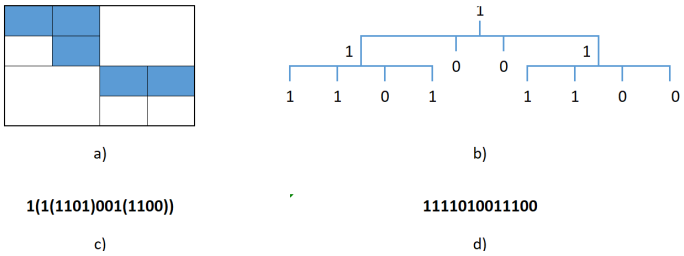


Figure 1

A 2D-space (for simplification of the example) discretized into 4 × 4 picture elements: a) with active pixels marked in blue; b) depicted as a quadrant tree; c) depicted in binary representation, with the decomposition of the respective quadrants indicated using parentheses; d) the final binary representation of the quadrant tree

3.2 Sparse Voxel Octrees

When creating SVO HDSs with pointers to existing child nodes, the binary representation of an array of pointers (PTS) is concatenated in the parent node after its CHNM. This array contains pointers (PTs) to the existing child nodes of the parent node. Every active child node has its own associated HT with a value of 1 in the CHNM of the parent node. The order of the PTs in the PTS is the same as the order of the HTs with value 1 in the CHNM. Each INODE must have at least one child node and may have a maximum of eight. Therefore, the number of PTs in the PTS is variable, and is from the interval <1;8>.

To formalize the binary representation of SVOs, we used the Backus-Naur Form:

SVO ::= (n)<NODE>

NODE ::= <INODE>|<LNODE>

INODE ::= <CHNM>(p)<BIT><PTS>

LNODE ::= <CHNM>(q)<BIT>

CHNM ::= (8)<HT>

PTS ::= (1)*(8)<PT>

PT ::= (r)<BIT>

HT ::= <BIT>

BIT ::= "0" | "1"

The BNF above uses the parameters p, r, and q. The parameters p and q determine the number of reserved bits that are added to align the CHNM to the desired number of bits. Parameter p is used for INODEs, parameter q for LNODEs. The parameter r determines the number of bits used to encode a pointer to a child node. By selecting these parameters correctly, one can align the size of parts of nodes or even whole nodes. For example, by setting the parameters as p = 24, q = 24, and r = 32, one can align all node parts and entire nodes to 32 bits. Figure 2 illustrates an example of a data structure setup (for the purpose of clarity, we used 2D).

The clear advantage of using pointers to child nodes is that they allow for quick traversal of the SVO. The disadvantage is that the binary representation of the internal nodes of the data structure will be significantly larger. This can be calculated as the ratio between the length of binary representation of the internal node of the SVO and the PSVO, according to the following equation:

$$ratio = \frac{SVOINODE_{size}}{PSVOINODE_{size}} = \frac{p + SVOCHNM_{size} + n \times PT_{size}}{PSVOCHNM_{size}} = \frac{p + 8 + n \times r}{8} \qquad (1)$$

where n is the number of PTs in the PTS.

For the above=mentioned setting where $p = 24$, $q = 24$ and $r = 32$, the SVOINODEsize is 64b if there is one child node. This value can grow to 288b if there are 8 child nodes. There is an 8-to-36 times increase in the size of SVO internal nodes compared to PSVO internal nodes. Even in a theoretical minimalist setting, which would not be sufficient for representing 3D scenes where $p = 0$, $q = 0$ and $r = 8$ (aligned to 8b), this increase is in the range of 2 to 9 times.

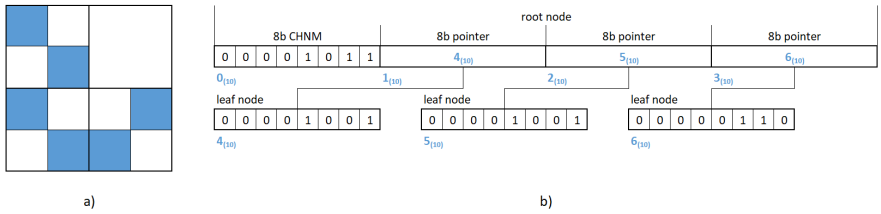


Figure 2

An example of encoding a 2D space into an SVO, with the parameters set to $p = q = 0$ and $r = 8$, where (a) is a 4 × 4 pixel 2D-scene; (b) is a binary representation of the SVO with addresses marked in blue and represented in decimal notation for simplicity and better visualization

The increase in the size of the SVO leaf node in comparison to the PSVO leaf node, can be calculated according to the following equation:

$$ratio = \frac{SVOLNODE_{size}}{PSVOLNODE_{size}} = \frac{q + SVOCHNM_{size}}{PSVOCHNM_{size}} = \frac{q + 8}{8} \qquad (2)$$

If the parameter q is set to 24, the size of the binary representation of the SVO leaf node is up to four times larger in comparison to PSVO. Conversely, if q is set to 0, the sizes of their binary representations are identical.

4 Dimensionality Scaling Principle

Pixels of 2D image or voxels of 3D image arranged in a regular 2D or 3D grid, respectively, as well as any data arranged in an n-dimensional regular grid, can be rearranged into a linear (1D) form. This can be achieved through the use of Space Filling Curves (SFCs). A prominent example of an SFC used in the field of computer graphics is the Morton Space Filling Curve (MSFC), also referred to as the Morton order or Z-order. This was initially proposed by G. M. Morton [24] in 1966. Another prevalent SFC is the Hilbert Space Filling Curve (HSFC), which was initially proposed by D. Hilbert in 1935 [25].

For images composed of voxels stored in a regular three-dimensional grid with a dimension of $N \times N \times N$ voxels, where $N = 2^m, m \geq 1, N, m \in \mathbb{N}$, each voxel has three coordinates, X, Y and Z, while $X \in < 0; 2^m - 1 >, Y \in < 0; 2^m - 1 >$ and $Z \in < 0; 2^m - 1 >, X, Y, Z \in \mathbb{N} +$. From these three coordinates, it is possible to construct the so-called Morton coordinate M by interleaving bits of the coordinates, with $M \in < 0; 2^{3m} - 1 >, M \in \mathbb{N} +$. This Morton coordinate then determines the position of the voxel in the linearized (1D) form of the image information.

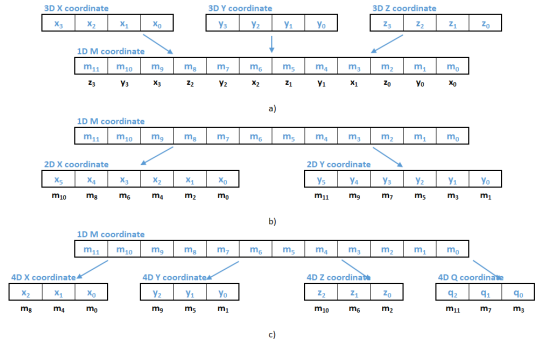


Figure 3
Transformation of coordinates: a) X, Y, Z 3D to M 1D, b) M 1D to X, Y 2D and c) M 1D to X, Y, Z, Q 4D

As an illustration, consider the construction of a Morton coordinate for a voxel that is part of a $16 \times 16 \times 16$ voxel image. Each voxel coordinate is composed of four bits, specifically X(3:0), Y(3:0), and Z(3:0). The Morton coordinate M is constructed from the 12 bits of M(11:0). This process is depicted in Figure 3(a).

Similarly, the inverse procedure can also be implemented, whereby the Morton coordinate of a specific voxel, representing its position in the linearized voxel stream, is converted to n coordinates to obtain the position of that voxel in n-dimensional space. It is possible to determine the number of dimensions in this space, which is given by the value of n, where $n \in \mathbb{N}$. Figure 3(b) demonstrates how a 12-bit Morton coordinate is transformed into two 6-bit *X*, *Y* coordinates of 2D space. Figure 3(c) illustrates the transformation of the same 12-bit Morton coordinate into 3-bit *X*, *Y*, *Z*, *Q* coordinates of 4D space.

If n and m represent the dimensionality of nD and mD space, respectively, it is possible to demonstrate the transformation of the n coordinates of the picture element from nD space to the $1D$ Morton coordinate $M\ (nD \rightarrow 1D)$ and subsequently, the 1D Morton coordinate can be transformed to the m coordinates of mD space $(1D \rightarrow mD)$. This process may be interpreted as a change in dimensionality from nD to $mD (nD \rightarrow mD)$.

Just as one can construct an octant tree (SVO) to a 3D grid of voxels storing information about 3D scene geometry, in a manner analogous to this, a quadrant tree may be constructed for 2D space, a binary tree for 1D space, a hexadeca tree for 4D space, and so on. Thus, it can be posited that an n-dimensional Sparse Voxel Tree (nD SVT) can be constructed for an nD grid. In order to formalize the binary representation of nD SVTs, we used the BNF:

SVT ::= (n) <NODE>
NODE ::= <INODE> | <LNODE>
INODE ::= <CHNM>(p) <BIT><PTS>
LNODE ::= <CHNM>(q) <BIT>
CHNM ::= (2^D)<HT>
PTS ::= (1)*(2^D)<PT>
PT ::= (r)<BIT>
HT ::= <BIT>
BIT ::= "0" | "1"

Where D is the dimensionality of the SVT.

For INODEs of such SVTs, the size of their CHNM equals to 2^D, and the number of pointers to child nodes can be in the range of < 1; 2^D >. For LNODEs, the size of their CHNM is 2^D. This value is doubled by increasing D by 1. Consequently, if a 3D SVT node has a CHNM size of 8b, a 4D SVT node has a CHNM size of 16b.

Considering a scene in nD, having a dimensionality of n, transformed into a scene in mD having a dimensionality of m, then the following can be said:

- if $n > m$, we refer to the transformation as dimensionality downscaling,
- while if $n < m$, we refer to the transformation as dimensionality upscaling.

If the scene geometry is represented by an nD SVT with a corresponding dimensionality of n and the target is a representation of the same scene by an mD SVT, with a corresponding dimensionality of m, then the dimensionality of the SVT is reduced (downscaled) if $n > m$ and increased (upscaled) if $n < m$, respectively.

If the representation of the scene geometry in mD space is created by transforming a representation of the scene geometry in nD space $(nD \rightarrow mD)$, then if g_n is the voxel geometry information of a particular voxel from the nD space, where the Morton coordinate of the particular voxel is M_n and g_m is the voxel geometry information of the voxel from mD space, where the Morton coordinate of the voxel is M_m, and it is assumed that $M_n = M_m$, then $g_n = g_m$. Thus, if a voxel v_n with a

Morton coordinate M in nD space is passive, the voxel v_m with the same Morton coordinate M in mD space will also be passive. Conversely, if v_n is active, v_m will also be active. By traversing the related nD SVT and mD SVT using the same Morton coordinate specification for the target voxel in both cases allows for the acquisition of identical information about its geometry.

In the case of nD SVTs with different dimensionalities, which are equivalent representations of the geometry of the same 3D scene, the binary representations of these structures usually have different sizes. We may attempt to find the optimal number of dimensions of the SVT that represents the geometry of a given scene with a minimum binary size. The next section of the paper deals with this search in the case of scenes generated by voxelizing polygonal surface models.

5 Test Results and Discussion

This section of the paper summarizes results of tests. Subsection 5.1 describes the respective scenes used for testing. Subsection 5.2 summarizes the results obtained in terms of the size of the nD SVT binary representation for the respective models, 3D scene sizes and degrees of dimensionality upscaling. Subsection 5.3 discusses the sources of the compression gains.

5.1 Datasets

Tests were performed using three 3D polygonal surface models composed of triangles that were voxelized into six scenes (resolutions were ranging from 128^3 to 4096^3). This process resulted into 18 test scenes. For each scene, active voxels were then extracted, their coordinates were transformed into 64b Morton addresses and those were sorted in ascending order and stored in a file. Finally, list of active voxels was then transformed into nD SVT HDS with different dimensionality for respective scene. The detailed parameters of the respective 3D scenes are shown in Table 1; their visualizations are depicted in Figure 4. Tests were performed using NVIDIA GeForce RTX 3060, 12 GB GDDR6.

Table 1

Characteristics of the 3D scenes created by embedding polygonal surface models stored in the WaveFront Technologies OBJ geometry definition file format into these scenes and then voxelizing them to various resolutions. Table summarizes the total number of voxels in the scenes, the number of active voxels and their percentage considering the total number of voxels, for each model and resolution.

	Resolution					
Resolution	128^3	256^3	512^3	1024^3	2048^3	4096^3
Voxels [10^6]	2	16	128	1024	8192	65,536

Angel Lucy—488,880 triangles						
Active voxels [10^3]	22.48	91.52	366.58	1453.10	5685.86	21,656.43
[%]	1.07	0.55	0.27	0.14	0.07	0.03
Skull—80,016 triangles						
Active voxels [10^3]	74.10	298.85	1192.04	4688.08	17,958.71	64,608.51
[%]	3.53	1.78	0.89	0.44	0.21	0.09
Porsche—22,011 triangles						
Active voxels [10^3]	54.20	233.04	969.11	3938.35	15,539.54	58,673.98
[%]	2.58	1.39	0.72	0.37	0.18	0.09

Figure 4
Visualization of the voxelized scenes used for testing purposes: (**a**) "Angel Lucy" at 512^3; (**b**) "Skull" at 512^3; (c) "Porsche" at 512^3.

5.2 Test Results

In the tests, the geometries of the Angel Lucy, Porsche, and Skull scenes were encoded into SVTs of varying dimensionality, ranging from three dimensions (3D SVT, equivalent to SVO) to eight dimensions (8D SVT). In the first set of tests, for both internal and leaf nodes of the SVT, we maintained the alignment of all parts of the nodes, and thus of the nodes as a whole, at a length of 32b across all versions of SVTs, from 3D SVTs to 8D SVTs. Pointers to child nodes in all SVTs – as far as their dimensionality is concerned – always had a binary representation length of 32b. The CHNM length of both internal and leaf nodes had to be aligned.

The CHNM size of the internal node as well as the size of the leaf node was set to 8b in the case of 3D SVT. Adding 24 reserved bits was used to make them match 32b. The CHNM size of the internal node as well as the size of the leaf node was set to 16b in the case of 4D SVT. Adding 16 reserved bits was used to make them match 32b. For SVTs with dimensionality from 5D to 8D, their internal node CHNM lengths as well as leaf node lengths are already naturally aligned to 32b as their lengths are 32b, 64b, 128b, and 256b, respectively, thus no reserved bits are used in the construction of the nodes, both internal and leaf nodes.

Table 2 shows, for each model, scene resolution and dimensionality of the nD SVT, the size of the binary representation of the corresponding nD SVT in KB and the percentage of the size of this binary representation compared to the size of the binary representation of the uncompressed 3D scene geometry (1b/vox 3D grid).

Table 3 shows, for each model, scene resolution and dimensionality of the nD SVT the data related to the compression ratios, with two values given each time. The first value represents the ratio between the compression ratios obtained when coding the nD SVT in the corresponding dimensionality n and in a lower dimensionality, i.e. $n-1$, e.g. 6D SVT and 5D SVT. The second value represents the ratio between the compression ratios obtained when coding the nD SVT in the corresponding dimensionality n and in 3D, e.g. 6D SVT and 3D SVT. The 3D SVT column always has a value of 1.00. Figure 5 shows ratio of the binary representation size of the nD SVT to the 3D SVT, for particular models and resolutions.

As it showed in the results obtained, for a given model and scene resolution, it is always true that with an increase in the dimensionality of the SVT HDS, the size of the binary representation of this data structure gradually decreased until it reached a minimum and then it increased again. Thus, an optimal SVT dimensionality can be found where the size of the SVT binary representation is the smallest and thus the compression ratio is the highest in comparison to SVO (equivalent to 3D SVT). This optimum was found in this set of tests for 6D SVT and 7D SVT, respectively, when the compression ratio in a given dimensionality compared to the compression ratio in 3D SVT ranged from 3.23 when using 6D SVO for a scene containing the Skull model with a resolution of 4096^3 to 4.44 when using 7D SVO for a scene containing the Porsche model with a resolution of 128^3.

Table 2
Size of the binary representation of the nD SVT in KB for test models and its ratio to the size of the geometry representation in the 1b/vox 3D grid. Node components are aligned to 32b.

Angel Lucy	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	53	31	17	15	16	16
	20.87%	12.22%	6.76%	5.79%	6.24%	6.40%
256^3	229	135	76	65	70	72
	11.18%	6.61%	3.69%	3.19%	3.42%	3.51%
512^3	944	567	318	278	302	317
	5.76%	3.46%	1.94%	1.70%	1.85%	1.93%
1024^3 ($1K^3$)	3808	2301	1297	1138	1259	1329
	2.91%	1.76%	0.99%	0.87%	0.96%	1.01%
2048^3 ($2K^3$)	15600	9201	5204	4574	5088	5386
	1.49%	0.88%	0.50%	0.44%	0.49%	0.51%
4096^3 ($4K^3$)	59562	36342	20615	18166	20308	21558
	0.71%	0.43%	0.25%	0.22%	0.24%	0.26%

Porsche	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	117	64	36	29	26	29
	45.64%	24.96%	14.21%	11.26%	10.29%	11.14%
256^3	540	300	181	147	145	156
	26.38%	14.65%	8.85%	7.15%	7.10%	7.60%

512^3	2361	1324	814	664	671	775
	14.41%	8.08%	4.97%	4.05%	4.10%	4.73%
1024^3 ($1K^3$)	9932	5605	3489	2877	2920	3440
	7.58%	4.28%	2.66%	2.20%	2.23%	2.62%
2048^3 ($2K^3$)	40700	23131	14507	12021	12296	14692
	3.88%	2.21%	1.38%	1.15%	1.17%	1.40%
4096^3 ($4K^3$)	162103	92904	58917	49030	50521	60923
	1.93%	1.11%	0.70%	0.58%	0.60%	0.73%

Skull	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	186	113	63	54	58	61
	72.68%	44.03%	24.80%	21.02%	22.73%	23.88%
256^3	765	467	264	225	251	266
	37.35%	22.78%	12.89%	11.00%	12.26%	12.97%
512^3	3100	1899	1079	922	1033	1101
	18.92%	11.59%	6.59%	5.63%	6.31%	6.72%
1024^3 ($1K^3$)	12413	7633	4353	3727	4199	4484
	9.47%	5.82%	3.32%	2.84%	3.20%	3.42%
2048^3 ($2K^3$)	49038	30311	17344	14891	16847	18067
	4.68%	2.89%	1.65%	1.42%	1.61%	1.72%
4096^3 ($4K^3$)	189341	118228	68027	58666	66782	71887
	2.26%	1.41%	0.81%	0.70%	0.80%	0.86%

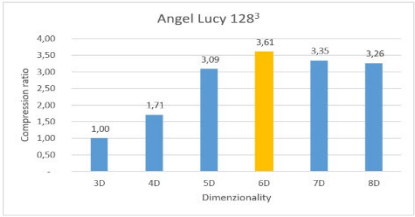
Table 3

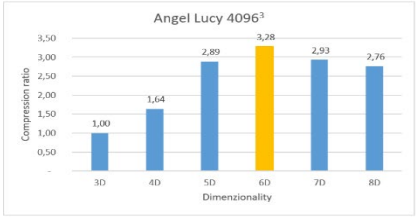
The ratio of the size of the nD SVT to the n-1D SVT and the ratio of the size of the nD SVT to the 3D SVT, for test models. Node components are aligned to 32b.

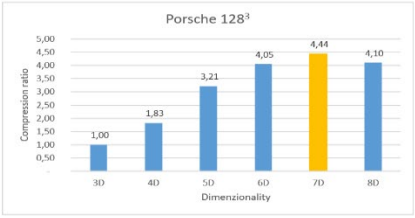
Angel Lucy	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	1.00	1.71	1.81	1.17	0.93	0.98
	1.00	1.71	3.09	3.61	3.35	3.26
256^3	1.00	1.69	1.79	1.16	0.93	0.97
	1.00	1.69	3.03	3.50	3.27	3.19
512^3	1.00	1.66	1.78	1.14	0.92	0.95
	1.00	1.66	2.96	3.39	3.12	2.98
1024^3 ($1K^3$)	1.00	1.65	1.77	1.14	0.90	0.95
	1.00	1.65	2.94	3.35	3.02	2.87
2048^3 ($2K^3$)	1.00	1.70	1.77	1.14	0.90	0.94
	1.00	1.70	3.00	3.41	3.07	2.90
4096^3 ($4K^3$)	1.00	1.64	1.76	1.13	0.89	0.94
	1.00	1.64	2.89	3.28	2.93	2.76

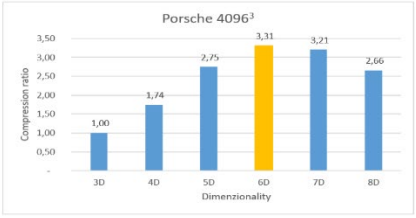
Porsche	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	1.00	1.83	1.76	1.26	1.09	0.92
	1.00	1.83	3.21	4.05	4.44	4.10
256^3	1.00	1.80	1.65	1.24	1.01	0.93
	1.00	1.80	2.98	3.69	3.72	3.47
512^3	1.00	1.78	1.63	1.23	0.99	0.87
	1.00	1.78	2.90	3.56	3.52	3.05
1024^3 ($1K^3$)	1.00	1.77	1.61	1.21	0.99	0.85
	1.00	1.77	2.85	3.45	3.40	2.89
2048^3 ($2K^3$)	1.00	1.76	1.59	1.21	0.98	0.84
	1.00	1.76	2.81	3.39	3.31	2.77
4096^3 ($4K^3$)	1.00	1.74	1.58	1.20	0.97	0.83
	1.00	1.74	2.75	3.31	3.21	2.66

Skull	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	1.00	1.65	1.78	1.18	0.92	0.95
	1.00	1.65	2.93	3.46	3.20	3.04
256^3	1.00	1.64	1.77	1.17	0.90	0.95
	1.00	1.64	2.90	3.39	3.05	2.88
512^3	1.00	1.63	1.76	1.17	0.89	0.94
	1.00	1.63	2.87	3.36	3.00	2.81
1024^3 ($1K^3$)	1.00	1.63	1.75	1.17	0.89	0.94
	1.00	1.63	2.85	3.33	2.96	2.77
2048^3 ($2K^3$)	1.00	1.62	1.75	1.16	0.88	0.93
	1.00	1.62	2.83	3.29	2.91	2.71
4096^3 ($4K^3$)	1.00	1.60	1.74	1.16	0.88	0.93
	1.00	1.60	2.78	3.23	2.84	2.63









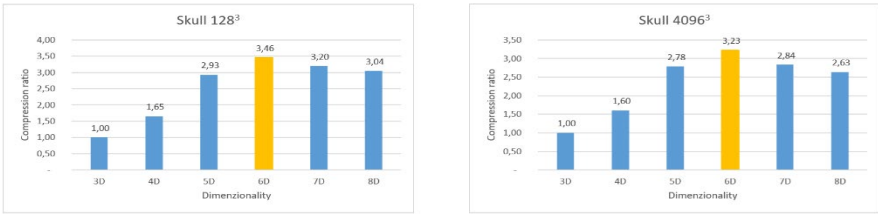


Figure 5

Ratio of the binary representation size of the nD SVT to the 3D SVT for the Angel Lucy, Porsche and Skull model, with optimal dimensionality indicated. Node components not aligned to 32b.

In the second set of tests, the same models and the same scene sizes were used. However, the alignment to 32b was not maintained for each of the SVT dimensionalities. Reserved bits were not used. This was manifested in the case of 3D and 4D SVTs, where the child node mask size and leaf node size were 8b and 16b for 3D and 4D SVTs, respectively. This allowed for a 37.5 percent and 25 percent reduction in the size of the binary representation of the 3D and the 4D SVTs, respectively, compared to the version with a 32b alignment. For 5D to 8D dimensionality, the SVT nodes are naturally aligned to 32b without using any reserved bits, as in the first set of tests. Therefore, there was no change in the size of the binary representation of the 5D to 8D SVTs in the second set of tests.

Table 4 shows the size of binary representation of the nD SVT in KB and the percentage of the size of this binary representation compared to the size of the binary representation of the uncompressed 3D scene geometry (1b/vox 3D grid).

Table 5 shows the data related to the compression ratios, with two values given each time. The first value represents the ratio between the compression ratios obtained when coding the SVT in the corresponding dimensionality n and in a lower dimensionality, i.e. $n-1$, e.g. 6D SVT and 5D SVT. The second value represents the ratio between the compression ratios obtained when coding the nD SVT in the corresponding dimensionality n and in 3D, e.g. 6D SVT and 3D SVT.

Table 4

Size of the binary representation of the nD SVT in KB for test models and its ratio to the size of the geometry representation by 1b/vox 3D grid. Node components are not aligned to 32b.

Angel Lucy	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	33	23	17	15	16	16
	13.04%	9.17%	6.76%	5.79%	6.24%	6.40%
256^3	143	102	76	65	70	72
	6.99%	4.96%	3.69%	3.19%	3.42%	3.51%
512^3	590	426	318	278	302	317
	3.60%	2.60%	1.94%	1.70%	1.85%	1.93%
1024^3 ($1K^3$)	2380	1726	1297	1138	1259	1329
	1.82%	1.32%	0.99%	0.87%	0.96%	1.01%

2048^3 ($2K^3$)	9475	6901	5204	4574	5088	5386
	0.90%	0.66%	0.50%	0.44%	0.49%	0.51%
4096^3 ($4K^3$)	37238	27257	20615	18166	20308	21558
	0.44%	0.32%	0.25%	0.22%	0.24%	0.26%

Porsche	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	73	48	36	29	26	29
	28.53%	18.72%	14.21%	11.26%	10.29%	11.14%
256^3	338	225	181	147	145	156
	16.49%	10.99%	8.85%	7.15%	7.10%	7.60%
512^3	1476	993	814	664	671	775
	9.01%	6.06%	4.97%	4.05%	4.10%	4.73%
1024^3 ($1K^3$)	6208	4203	3489	2877	2920	3440
	4.74%	3.21%	2.66%	2.20%	2.23%	2.62%
2048^3 ($2K^3$)	25438	17348	14507	12021	12296	14692
	2.43%	1.65%	1.38%	1.15%	1.17%	1.40%
4096^3 ($4K^3$)	101314	69678	58917	49030	50521	60923
	1.21%	0.83%	0.70%	0.58%	0.60%	0.73%

Skull	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	116	85	63	54	58	61
	45.42%	33.02%	24.80%	21.02%	22.73%	23.88%
256^3	478	350	264	225	251	266
	23.35%	17.08%	12.89%	11.00%	12.26%	12.97%
512^3	1937	1424	1079	922	1033	1101
	11.82%	8.69%	6.59%	5.63%	6.31%	6.72%
1024^3 ($1K^3$)	7758	5725	4353	3727	4199	4484
	5.92%	4.37%	3.32%	2.84%	3.20%	3.42%
2048^3 ($2K^3$)	30649	22734	17344	14891	16847	18067
	2.92%	2.17%	1.65%	1.42%	1.61%	1.72%
4096^3 ($4K^3$)	118338	88671	68027	58666	66782	71887
	1.41%	1.06%	0.81%	0.70%	0.80%	0.86%

Table 5

Size of the binary representation of the nD SVT in KB for test models and its ratio to the size of the geometry representation in the 1b/vox 3D grid. Node components are not aligned to 32b.

Angel Lucy	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	1.00	1.42	1.36	1.17	0.93	0.98
	1.00	1.42	1.93	2.25	2.09	2.04
256^3	1.00	1.41	1.34	1.16	0.93	0.97
	1.00	1.41	1.89	2.19	2.04	1.99

512^3	1.00	1.39	1.34	1.14	0.92	0.95
	1.00	1.39	1.85	2.12	1.95	1.86
1024^3 ($1K^3$)	1.00	1.38	1.33	1.14	0.90	0.95
	1.00	1.38	1.84	2.09	1.89	1.79
2048^3 ($2K^3$)	1.00	1.37	1.33	1.14	0.90	0.94
	1.00	1.37	1.82	2.07	1.86	1.76
4096^3 ($4K^3$)	1.00	1.37	1.32	1.13	0.89	0.94
	1.00	1.37	1.81	2.05	1.83	1.73

Porsche	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	1.00	1.52	1.32	1.26	1.09	0.92
	1.00	1.52	2.01	2.53	2.77	2.56
256^3	1.00	1.50	1.24	1.24	1.01	0.93
	1.00	1.50	1.86	2.30	2.32	2.17
512^3	1.00	1.49	1.22	1.23	0.99	0.87
	1.00	1.49	1.81	2.22	2.20	1.90
1024^3 ($1K^3$)	1.00	1.48	1.20	1.21	0.99	0.85
	1.00	1.48	1.78	2.16	2.13	1.80
2048^3 ($2K^3$)	1.00	1.47	1.20	1.21	0.98	0.84
	1.00	1.47	1.75	2.12	2.07	1.73
4096^3 ($4K^3$)	1.00	1.45	1.18	1.20	0.97	0.83
	1.00	1.45	1.72	2.07	2.01	1.66

Skull	3D SVT	4D SVT	5D SVT	6D SVT	7D SVT	8D SVT
128^3	1.00	1.38	1.33	1.18	0.92	0.95
	1.00	1.38	1.83	2.16	2.00	1.90
256^3	1.00	1.37	1.33	1.17	0.90	0.95
	1.00	1.37	1.81	2.12	1.90	1.80
512^3	1.00	1.36	1.32	1.17	0.89	0.94
	1.00	1.36	1.79	2.10	1.88	1.76
1024^3 ($1K^3$)	1.00	1.36	1.32	1.17	0.89	0.94
	1.00	1.36	1.78	2.08	1.85	1.73
2048^3 ($2K^3$)	1.00	1.35	1.31	1.16	0.88	0.93
	1.00	1.35	1.77	2.06	1.82	1.70
4096^3 ($4K^3$)	1.00	1.33	1.30	1.16	0.88	0.93
	1.00	1.33	1.74	2.02	1.77	1.65

Figure 6 shows ratio of the binary representation size of the nD SVT to the 3D SVT, for particular models and resolutions. The aforementioned significant size reduction of the 3D and 4D SVT binary representation was not sufficient to make these data structures more compact than the 5D SVT, as can be seen from the test results. Thus,

similarly to the first set of tests, for a given model and scene resolution, with an increase in the dimensionality of the SVT, the size of the binary representation of this data structure gradually decreased until it reached a minimum and then it increased again.

An optimal SVT dimensionality can be found where the size of the SVT binary representation is the smallest. This optimum was also found in this test set for the 6D and 7D SVTs, respectively, as it was found in the first test set. However, compared to the first test set, we achieved a lower compression ratio in this dimensionality than in the case of the 3D SVT (which corresponds to the SVO). This is due to the smaller size of the binary representation of the 3D SVT in this second set of tests. It ranged from 2.02 when using the 6D SVT for the scene containing the Skull model with a resolution of 4096^3 to 2.77 when using the 7D SVT for the scene containing the Porsche model with a resolution of 128^3.

Figure 6

Ratio of the binary representation size of the nD SVT to the 3D SVT for the Angel Lucy, Porsche and Skull model, with optimal dimensionality indicated. Node components not aligned to 32b.

6 Discussion

Increasing the dimensionality of a 3D scene results in two opposite tendencies when generating nD SVTs. The negative tendency is that as the dimensionality increases, the CHNM size of both internal and leaf nodes increases sharply, with the CHNM size of a node doubling for every increase in dimensionality by 1. Thus, the 256b CHNM of 8D SVT has a binary representation up to 32 times larger than the 8b CHNM of 3D SVT. The positive tendency is that as the dimensionality increases, the need to use reserved bits to align parts of the data structure nodes disappears in SVTs with higher dimensionalities, which means a reduction in the size of the binary representation by 24b compared to 3D and by 16b compared to 4D SVTs for each node, when aligning both internal and leaf nodes to 32b. At the same time, the reduction in the total number of nodes, and thus the number of pointers to particular child nodes, is also a positive trend. Every node u (except the root node) of the data structure has a parent node p. The parent node has a pointer to this child node u. Thus, reducing the number of nodes by 1 means reducing the size of the binary representation of the data structure by the size of a child node mask and the size of a pointer to the child node.

The size of the data structure $nDSVT_{size}$ can then be expressed as

$$nDSVT_{size} = (n+m) \times CHNM_{size} + (n+m-1) \times PT_{size} \quad [b] \tag{2}$$

where

n is the number of internal nodes of the data structure

m is the number of leaf nodes of the data structure

$CHNM_{size}$ is the size of the child node mask in bits

PT_{size} is the size of the child node pointer in bits

The optimal dimensionality of the nD SVT then becomes the one where these above-mentioned opposing tendencies - increase in CHNM size and reduction of the number of nodes and pointers to nodes - reach balance.

All node components of the data structure are naturally aligned to 32 bits from the 5D SVT. This satisfies the basic requirement for creating a GPU-friendly data structure. Traversing an nD SVT has linear time complexity $O(n)$, where n is the number of nodes to be traversed. The number of operations required to process a single node remains the same regardless of SVT dimensionality.

However, as dimensionality increases, the speed of node traversal is negatively impacted by the rapidly growing size of the CHNM (especially in 7D and 8D, where the CHNM length exceeds 64 bits) and by the increasing number of child pointers in the pointer array. As a result, the probability of cache misses increases. On the other hand, higher-dimensional SVTs require fewer levels to represent the same scene, thus reducing the total number of nodes that must be traversed from the root of the HDS to a leaf node.

Table 6 shows the traversal times (in µs) for all three tested models voxelized at a resolution of 1024^3 voxels. Traversal was performed in parallel for all active voxels in the scene, from the root of the nD SVT to their respective leaf nodes using NVIDIA GeForce RTX 3060 graphics card.

Table 7 and Figure 7 present the ratios of traversal times between 3D and higher-dimensional SVTs for the same model. In all tests, the 6D SVT proved to be the most efficient: traversal speed improved from 3D up to 6D, while in 7D and 8D, the need to process entities larger than 64 bits had a negative impact.

Table 6

nD SVT traversing time in µs for all active voxels in the scene with resolution of 1024^3 vox

Time [µs]	**3D**	**4D**	**5D**	**6D**	**7D**	**8D**
Angel Lucy 1024^3	99.4	89.1	75.8	73.7	158.8	244.7
Porsche 1024^3	257.0	225.3	198.7	190.5	428.0	654.3
Skull 1024^3	313.4	275.5	240.7	231.4	486.4	777.2

Table 7

Ratio between 3D and nD SVT traversing time for model voxelized to 1024^3 vox. resolution

Ratio	**3D**	**4D**	**5D**	**6D**	**7D**	**8D**
Angel Lucy 1024^3	1.00	1.12	1.31	1.35	0.63	0.41
Porsche 1024^3	1.00	1.14	1.29	1.35	0.60	0.39
Skull 1024^3	1.00	1.14	1.30	1.35	0.64	0.40

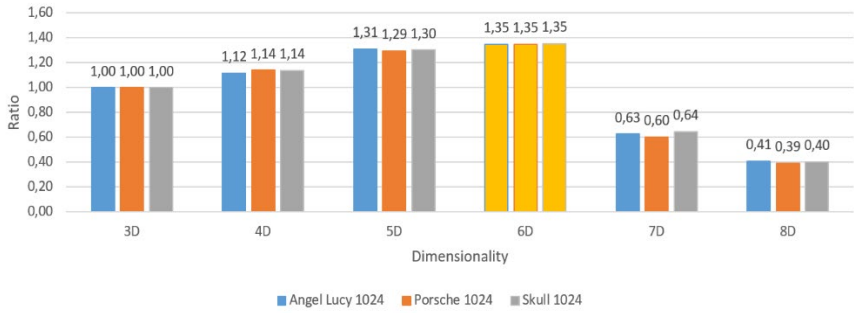


Figure 7

Ratio between nD and 3D SVT traversing time for model voxelized to 1024^3 vox. resolution

Another potential direction for future research in this field is the application of dimensionality upscaling in combination with CSM, which will result in an nD SVDAG HDS. More robust nodes of nD SVT can have negative impact on the successful use of CSM. An alternative approach is the use of FBC, where multiple pointer lengths to child nodes are introduced, potentially reducing the memory footprint required for representing child node pointers. This approach assumes a negative impact of CHNM length doubling, which should be mitigated.

Conclusions

This paper has addressed the problem of representing the geometry of voxelized three-dimensional scenes through hierarchical data structures. It has explored the possibility of using dimensionality upscaling of 3D scenes when these scenes can be transformed into scenes with a higher number of dimensions. The subsequent representation of the geometry of these scenes in the nD SVT hierarchical data structure can help to achieve a smaller size of the binary representation of the nD SVT, compared to the classical SVO.

Tests on scenes generated by voxelizing initially polygonal surface models showed that the optimal results, i.e. the most compact binary representations, were obtained by upscaling the dimensionality to 6D and 7D, respectively. A 3.23 to 4.44 times more compact binary representation of the nD SVT compared to the classical three-dimensional SVO was obtained for the nD SVT, where all parts of the nodes were aligned to 32b for all dimensionality versions. For the nD SVT without aligning all parts of the nodes to 32b for all dimensionality versions, a 2.02 to 2.77 times more compact binary representation of the nD SVT was achieved in comparison to the classical three-dimensional SVO. The 6D proved to be the optimal dimensionality in terms of traversing speed.

In future research, related to dimensionality upscaling, the influence of other properties of hierarchical data structures – the use of techniques such as Common Subtree Merge or Frequency Based Compaction, or the use of symmetry and other transformations – on the achieved degree of compactness of the data structure is to be investigated. Special attention will be paid to the compaction of child node masks, as this becomes a limiting factor in nD SVTs in achieving a higher degree of compactness of its binary representation.

Acknowledgement

This work was supported by KEGA Agency of the Ministry of Education, Science, Research, and Sport of the Slovak Republic under Grant No. 015TUKE-4/2024 Modern Methods and Education Forms in the Cybersecurity Education.

References

[1] Hunter, G. M.; Steiglitz, K.: Operations on Images Using Quad Trees. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, 1979, No. 2, pp. 145-153, doi: 10.1109/TPAMI.1979.4766900

[2] Kawaguchi, E.; Endo, T.: On a Method of Binary Picture Representation and Its Application to Data Compression. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-2, 1980, No. 1, pp. 27-35, doi:10.1109/TPAMI.1980.4766967

[3] Webber, R. E.; Dillencourt, M. B.: Compressing Quadtrees via Common Subtree Merging. Pattern Recognition Letters, Vol. 9, 1989, No. 3, pp. 193-200, doi:10.1016/0167-8655(89)90054-8

[4] Chang, H. K.: C.—Liu, S.-H.; Tso, C.-K.: Two-Dimensional Template-Based Encoding for Linear Quadtree Representation. Photogrammetric Engineering and Remote Sensing, Vol. 63, 1997, No. 11, pp. 1275-1282

[5] Parker, E.; Udeshi, T.: Exploiting Self-Similarity in Geometry for Voxel Based Solid Modeling. Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications (SM '03), 2003, pp. 157-166, doi: 10.1145/781606.781631

[6] Srihari, S. N.: Representation of Three Dimensional Digital Images. Technical Report No. 162, Department of Computer Science, State University of New York at Bufallo, Amherst, New York, pp. 26, 1980

[7] Rubin, S. M.; Whitted, T.: A 3-Dimensional Representation for Fast Rendering of Complex Scenes. Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '80), ACM, 1980, pp. 110-116, doi: 10.1145/800250.807479

[8] Tanimoto, S. L.: OctTrees and Their Use in Representing Three-Dimensional Objects. Computer Graphics and Image Processing, Vol. 14, 1980, No. 3, pp. 249-270, doi: 10.1016/0146-664X(80)90055-6

[9] Laine, S.; Karras, T. Efficient Sparse Voxel Octrees-Analysis, Extensions, and Implementation, NVIDIA Technical Report NVR-2010-001, NVIDIA Corporation, Santa Clara, USA, 2010; p. 30

[10] Madoš, B.; Chovancová, E.; Chovanec, M.; Ádám, N. CSVO: Clustered Sparse Voxel Octrees—A Hierarchical Data Structure for Geometry Representation of Voxelized 3D Scenes. Symmetry 2022, 14, 2114, https://doi.org/10.3390/sym14102114

[11] Baert, J.; Lagae, A.; Dutré, Ph. Out-of-Core Construction of Sparse Voxel Octrees. In Proceedings of the 5th High-Performance Graphics Conference (HPG '13), Anaheim, CA, USA, 19-21 July, 2013, pp. 27-32, https://doi.org/10.1145/2492045.2492048

[12] Pätzold, M.; Kolb, A. Grid-free out-of-core voxelization to sparse voxel octrees on GPU. In Proceedings of the 7th Conference on High-Performance Graphics (HPG '15), Los Angeles, CA, USA, 7-9 August 2015, pp. 95-103, ISBN 9781450337076, https://doi.org/10.1145/2790060.2790067

[13] Kämpe, V.; Sintorn, E.; Assarson, U. High Resolution Sparse Voxel DAGs. ACM Transactions on Graphics 2013} 32, pp. 1-13, ISSN 0730-0301, https://doi.org/10.1145/2461912.2462024

[14] Villanueva, A. J.; Marton, F.; Gobetti, E. Symmetry-aware Sparse Voxel DAGs (SSVDAGs) for compression-domain tracing of high-resolution geometric scene. {\em J. Comput. Graph. Tech. (JCGT)} 2017, 6, p. 30, http://jcgt. org/ published/0006/02/01

[15] Vokorokos, L.; Madoš, B.; Bilanová, Z. PSVDAG: Compact Voxelized Representation of 3D Scenes Using Pointerless Sparse Voxel Directed Acyclic Graphs. Computing and Informatics, 2020, 39, pp. 587-616, ISSN 1335-9150 (print) https://doi.org/10.31577/cai_2020_3_587

[16] Madoš, B.; Ádám, N. Transforming Hierarchical Data Structures-A PSVDAG-SVDAG Conversion Algorithm. Acta Polytechnica Hungarica, 2021, 18, pp. 47-66, ISSN 1785-8860, doi.org/10.12700/ APH.18.8.2021.8.3

[17] van der Laan, R.; Scandolo, L.; Eisemann, E. Lossy Geometry Compression for High Resolution Voxel Scenes. Proc. of the ACM on Comp. Graph. and Inter. Techniques 2020, p. 13, ISSN 2577-6193, doi.org/10.1145/3384541

[18] Careil, V.; Billeter, M.; Eisemann, E. Interactively Modifying Compressed Sparse Voxel Representations. In Computer Graphics Forum, 2020, 39, pp. 111-119, ISSN 0167-7055, https://doi.org/10.1111/cgf.13916

[19] Dolonius, D.; Sintorn, E.; Kämpe,V.; Assarsson, U. Compressing Color Data for Voxelized Surface Geometry. IEEE Transactions on Visualization and Computer Graphics 2019, 25, pp. 1270-1282, ISSN 1077-2626, https://doi.org/10.1109/TVCG.2017.2741480

[20] Williams, R. B. Moxel DAGs: Connecting Material Information to High Resolution Sparse Voxel DAGs. Master Thesis, California Polytechnic State University, San Luis Obispo, USA, 2015, doi.org/10.15368/theses.2015.112

[21] Dado, B.; Timothy R. K.; Bauszat, P.; Thiery J.-M.; Eisemann, E. Geometry and Attribute Compression for Voxel Scenes. In Proceedings of the 37th Annual Conference of the European Association for Computer Graphics, Lisbon, Portugal, 9-13 May 2016; pp. 397-407

[22] Sintorn, E.; Kämpe, V.; Olsson, O.; Assarson U. Compact precomputed voxelized shadows. ACM Transactions on Graphics 2014, 33, p. 8, ISSN 0730-0301, https://doi.org/10.1145/2601097.2601221

[23] Kämpe, V.; Sintorn, E.; Assarson U. Fast, Memory-Efficient Construction of Voxelized Shadows. In Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, San Francisco, CA, USA, 27 Feb.-1 Mar. 2015; pp. 25-30, ISBN 978-1-4503-3392-4, doi.org/10.1145/2699 276.2699284

[24] Morton, G. M. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing, Research Report. International Business Machines Corporation (IBM), Ottawa, Canada, 1 March 1966; p. 20

[25] Hilbert, D. \Via the continuous mapping of a line onto a patch of area (Über die stetige Abbildung einer Linie auf ein Flächenstück). Dritter Band: Analysis Grundlagen der Mathematik Physik Verschiedenes; Springer, Germany, 1935; ISBN 978-3-662-37657-7; ISBN 978-3-662-38452-7