# Parameter Optimization in Sparse Voxel DAGs for Efficient Geometry Representation of Voxelized 3D Scenes

**Branislav Madoš, Norbert Ádám, Eva Chovancová, Heidar Khorshidiyeh and 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, norbert.adam@tuke.sk, eva.chovancova@tuke.sk, heidar.khorshidiyeh@tuke.sk, peter.poprik@tuke.sk

*Abstract: This paper addresses the issue of geometry representation of voxelized three-dimensional scenes, using domain-specific hierarchical data structures, utilizing lossless compression. It discusses the details of the sparse voxel directed acyclic graph hierarchical data structure with implemented child node pointers, including a detailed analysis of its construction. Then, based on this analysis, we discuss the possibilities of adjusting its parameters, in order to optimize it in terms of its binary representation size. In particular, the possibility of using various numbers of child node pointer types and the binary representation lengths of these is considered; this is also related to the different binary representation lengths of the child node mask and the different binary representation lengths and meanings of the constituent header tags. The various settings of the data structure parameters are then tested on three models whose original representation – surface polygonal models – was transformed: each model was voxelized to six different resolutions. The results obtained were also compared with an uncompressed representation of the geometry of the voxelized three-dimensional scene, encoded using a regular three-dimensional grid of one-bit scalar values (1b/vox). Based on these tests, using a 16b child node mask, three different child node pointer lengths (8b, 16b and 32b) and an 8b address word length came out as the best parameter combination of the investigated data structure in terms of data representation compactness. In the tests, a 1.88–3.06-fold data compression rate was achieved with this parameter configuration for the used test models and scene resolutions, compared to the traditional SVDAG structure using 32b pointers. The maximum data compression ratio (1098.3) – compared to the 3D grid of 1b/vox scalar values – was achieved with this parameter setting for a scene resolution of $4096^3$ voxels.*

*Keywords: voxelized three-dimensional scene; geometry of the scene; hierarchical data structure; sparse voxel octrees; sparse voxel directed acyclic graphs; common subtree merge; frequency-based compaction*

# 1 Introduction

In domains such as computer graphics or computer vision, representation of image data utilises a variety of different methods, with one such common approach being its encoding as regular multi-dimensional grids of image components. These components are termed *pixels* (picture elements) for two-dimensional (2D) images and *voxels* (volume picture elements) for three-dimensional (3D) images. The aforementioned image components are then assigned different attributes, e.g., colour or different material properties. Despite the simplicity of this approach, both in terms of data encoding and processing, storing image data in such a straightforward, uncompressed form requires a lot of storage space and is not considered efficient, for 2D and even so for 3D image data. The criticality of this problem can be illustrated by the example of encoding the geometry information of 3D voxelized scenes in computer graphics. The implementation of this approach involves the allocation of a single bit of information to each voxel within the scene, assigning a *0* value to empty (passive) voxels and *1* value to full (active) voxels. This approach, despite its relatively modest overhead, necessitates the allocation of up to 64 Gb (8GB) of operating memory, graphics card memory or secondary storage space for a scene with a resolution of $4096 \times 4096 \times 4096$ ($4K^3$) voxels. In light of this, both lossless and lossy compression algorithms, along with their respective data structures, have been identified as potential solutions.

Hierarchical data structures (HDSs), a subject of study for decades [1], have emerged as an effective solution. The use of quadtrees for the representation of 2D image data [2] has been proposed, while repeated patterns present in these data have been also the focus of investigation [3] [4]. Since the 1980s, there has been significant research interest in using equivalent HDSs for storing 3D image data, in the form of octrees [5-7]. The investigation continues today into their modern versions, which are designed for GPU-friendly representation of voxelized 3D scenes sparsely populated with active voxels (hence use of the *sparse* adjective in this context). Modern approaches are focused on various aspects, including scene geometry (see the Related Works for further details), material properties [8] [9], shadows [10] [11] and temporal scene dynamics [12]. In these HDSs, the iterative decomposition of multi-dimensional image information into quadrants (2D) and octants (3D), respectively, is reflected by a hierarchical decomposition of their nodes (i.e. the parent node) into four and eight potential nodes (i.e., child nodes), respectively. Quadrants (octants) can be classified as passive (composed entirely of passive pixels or voxels) and active (containing at least one active pixel or voxel). If the relationship between parent and child nodes in an HDS is encoded such that the parent node representation explicitly encodes the address of the child node, then the given HDS is classified as one having child node pointers implemented (for rapid traversal). If pointers are not implemented and the parent node – child node relationship is determined by their relative location within the HDS, it can be classified as a HDS without a pointer implementation, termed a *pointerless* HDS (suitable for data archiving or streaming).

The efficiency of these HDSs in representing the geometry of a 3D scene is contingent on the assumption that the scene is sparse, i.e., the ratio of active voxels, compared to their total number, is very small. In fact, they can represent passive octants, occurring with high frequency in sparse scenes, with great efficiency. This is directly reflected in the nomenclature of these HDSs. A tree-based HDS with pointers implemented is denoted a *Sparse Voxel Octree* (SVO). The development of the *Sparse Voxel Directed Acyclic Graph* (SVDAG) HDS involved the integration of a search and an economical representation of multiple occurrences of identical subtrees, termed *Common Subtree Merge* (CSM). Subsequent modifications to these HDSs included the incorporation of additional features, such as mirroring, the utilisation of multiple child-node-pointer lengths and the related *Frequency Based Compaction* (FBC) technique. This development has led to the creation of numerous HDSs (see the Related Works section of this paper).

This paper focuses on HDSs called Sparse Voxel Directed Acyclic Graphs (SVDAGs). It examines the potential for optimising the component parameters of its nodes, including the size of the binary representation and the encoding of the Child Node Mask (CHNM), as well as the number of different types of child node pointers, along with the length of their binary representation. Additionally, it explores the mutual influence of these parameters. The objective is to evaluate their impact on the overall size of the SVDAG binary representation when compressing the geometry of voxelized 3D scenes. This paper undertakes a theoretical analysis of implications of specific parameter settings and provides an empirical verification of these, using a sample of voxelized 3D scenes, before selecting the optimal parameter configuration.

The contribution of this paper primarily provides an analysis and testing of the parameter-setting options of SVDAG component nodes, with regard to their impact on the size of its binary representation; secondly, it identifies the optimal setting of these parameters.

This paper is structured in 6 sections. Section 2 focuses on the related works in the area of geometry representation of voxelized 3D scenes using HDSs. Section 3 deals with the analysis of the construction of SVO and SVDAG HDSs in terms of the construction of their internal as well as leaf nodes and highlights the source of SVDAG compactness compared to SVO. Section 4 discusses the possibilities, context and implications of different SVDAG parameter settings. These include the binary representation length and Child Node Mask encoding, as well as the number of different types of child node pointers and their lengths in relation to the size of the binary representation of this HDS. Section 5 summarises the results of the tests performed on different models voxelized to different resolutions and then stored in SVO and SVDAG with different parameter settings. Finally, Section 6 outlines the conclusions drawn, based on the content of the preceding sections of the paper.

# 2 Related Work

HDSs designed for representing the geometry of voxelized scenes, based on the use of trees, include the Sparse Voxel Octree (SVO). In 2013, Baert et al. proposed a two-step algorithm for SVO construction in [13]. The initial step involves processing a mesh of triangles into an intermediate product, which is a sorted list of active voxels. The subsequent step is to process this list into an SVO. In 2015, Pätzold and Kolb proposed a voxelization algorithm that also produces an SVO, eliminating the requirement for the creation of a memory-intensive intermediate product [14]. The concept of Efficient Sparse Voxel Octrees (ESVOs) was introduced by Laine and Karras in 2010. Its advantage over the traditional SVOs lies in the possibility of replacing entire subtrees of a data structure by contour information [15]. This is more economical in terms of its binary representation compared to the representation of the subtree being replaced. In 2022, a HDS was proposed by Madoš et al. in [16], in the form of Clustered Sparse Voxel Octrees (CSVOs). This structure has multiple types of internal nodes with differing encodings of Header Tags (HT) of child node masks; it also incorporates different child node pointer lengths.

HDSs designed for representing the geometry of voxelized 3D scenes, based on the use of directed acyclic graphs, include Sparse Voxel Directed Acyclic Graphs (SVDAGs), introduced by Kämpe et al. in 2013 [17]. In contrast to SVOs, SVDAGs allow multiple referencing of a child node from one or more parent nodes. This technique is termed Common Subtree Merge (CSM). It facilitates the straightforward and efficient optimization of the size of the data structure. An evolution of SVDAGs, Symmetry-aware Sparse Voxel Directed Acyclic Graphs (SSVDAGs) was introduced in 2016 by Villanueva et al. [18]. This concept introduces the possibility of employing CSM even when mirroring in one or more scene axes is necessary to achieve identity. It also introduces several different child node pointer lengths, thereby enabling Frequency Based Compaction (FBC). In 2020, Pointerless Sparse Voxel Directed Acyclic Graphs (PSVDAGs) were introduced by Vokorokos et al. in [19]. This HDS combines the advantages of the compactness of pointerless data structures with the possibility of implementing CSM, utilising a wide range of pointer lengths in combination with FBC. Subsequent to this proposal, Madoš and Ádám [20] proposed a conversion algorithm for transforming PSVDAGs into SVDAGs in 2021.

Lossy Sparse Voxel Directed Acyclic Graphs (LSVDAGs), as proposed by van der Laan in 2020 [21], is based on the SVDAG structure; however, this concept allows lossy data compression by modifying the subtrees of the data structure to increase the frequency of CSM usage. In 2020, Careil et al. presented a solution that allowed for interactive modification of the scene geometry information recorded in sparse voxel representations [22].

# 3 SVOs and SVDAGs

Hierarchical data structures can be used to encode the geometry of voxelized 3D scenes sized $N^3$ voxels, where $N = 2^m; m \in \mathbb{N}$. This condition is imposed to enable iterative decomposition of the scene into octants, by halving it in each of the three axes of the image, until reaching the level of individual voxels. The corresponding HDS is then composed of nodes embedded in $m$ layers, each of which can be denoted by a natural number $l \in < 0; m-1 >$. The layer with $l = 0$ stores the root node of the HDS, while the layer $l = m-1$ stores the leaf nodes. Nodes in layers $l \in < 0; m-2 >$ are constructed as Internal Nodes (INODEs), while nodes in layer $l = m-1$ are constructed as Leaf Nodes (LNODEs). For each decomposition into octants (iteration), eight octants are created and a node representing that decomposition is stored in the appropriate node layer of the HDS. The depth of this layer corresponds to the depth of the relevant iteration. If the resulting octant only contains passive voxels, it is considered passive and will not undergo further decomposition. This represents a significant economy in terms of the representation of the HDS. Conversely, if an octant contains at least one active voxel, it is classified as active and undergoes further iterative decomposition, along with the creation of a dedicated node. This iterative decomposition process ends when an active octant of $2^3$ voxels is produced. Subsequently, a leaf node carrying information regarding the passivity/activity of specific voxels is formed; this does not undergo further decomposition. INODEs are composed of a Child Node Mask (CHNM) and an array of child node pointers (PTS); LNODEs consist of a CHNM (they are not further decomposed and therefore do not contain an array of child node pointers).

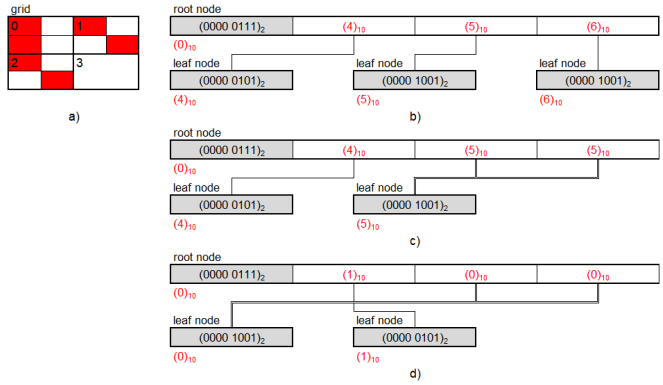


Figure 1

Representation of the scene geometry using: a) a 4×4 pixel 2D grid (red pixels are active); b) an SVO with 8b child node pointers; c) an SVDAG with 8b child node pointers using CSM; and d) an SVDAG with 8b child node pointers using CSM and FBC (CHNMs are depicted in grey)

In the case of an octant whose (parent) node is stored in layer $l$, this is decomposed into eight sub-octants. In the parent node, it is necessary to record which of these sub-octants is passive and thus will not be further decomposed, and which is active

and will have its (child) node stored in the $l + 1$ node layer. This information is recorded in the parent node in its Child Node Mask (CHNM), composed of eight Header Tags (HTs) – one for each of the sub-octants. The order of these is subject to agreement; for example, it can be determined using space filling curves such as the Morton Space Filling Curve (MSFC) or the Hilbert Space Filling Curve (HSFC). To facilitate efficient traversal of the data structure, parent nodes contain pointers to active child nodes (PT). These pointers to child nodes form an array (PTS), concatenated after the CHNM. The pointers are arranged in the same order as the HTs in the CHNM. It is imperative that each INODE possesses at least one active child node, thereby ensuring that at least one HT in its CHNM encodes information pertaining to the active child node. Consequently, the array of child node pointers must contain at least a single pointer. The number of active child nodes of a given parent node is variable, ranging from 1 to 8. This range applies also to the number of pointers to child nodes in the child node pointer array. The total size of the binary representation of the internal node, denoted $INODE_{len}$, can be calculated using the following formula:

$$INODE_{len} = r + 8 \times ht_{len} + n \times pt_{len} \text{ [b]} \tag{1}$$

Where,

$r$ is the number of bits allocated for CHNM alignment;

$ht_{len}$ is the number of bits constituting the Header Tag in the CHNM ($ht_{len} = 1$);

$n$ is the number of active child nodes and hence the number of pointers to them;

$pt_{len}$ is the number of bits forming a pointer to a child node.

For the size of the binary representation of a leaf node, denoted $LNODE_{len}$, the following formula applies:

$$LNODE_{len} = r + 8 \times ht_{len} \text{ [b]} \tag{2}$$

Where,

$r$ is the number of bits allocated for CHNM alignment;

$ht_{len}$ is the number of bits constituting the Header Tag in the CHNM ($ht_{len} = 1$)

In order to align the components of a node to the desired length of their binary representation, the CHNM is padded with a certain number of reserved bits. If the length of child node pointers is $pt_{len} = 32b$, then an 8b CHNM can be aligned to 32b by using the reserved bits, where $r = 24$. With this alignment, it is then possible to use 32b addressing word length within the HDS. The decomposition of a 2D space into quadrants (a 2D space was selected for the sake of simplicity) is illustrated in Figure 1a, where a $4 \times 4$ pixel grid is shown and the space is iteratively decomposed into 4 quadrants. Due to the modest proportions of the grid, a solitary decomposition into 4 quadrants of size $2 \times 2$ voxels were executed; the quadrants thus formed already have a minimum dimension of $2^2$ pixels and are therefore not decomposed any further). Figure 1b depicts the corresponding SVO containing a

solitary root node, consisting of a grey-labelled CHNM and three 8b PT pointers to the child nodes in the PTS. The child nodes are constructed as LNODEs and thus their CHNM is depicted in grey. This CHNM contains the information about the active and passive voxels of the quadrant; however, it holds no pointers to further child nodes.

Table 1

Number of nodes of the SVO and SVDAG hierarchical data structure and their ratio for the Lucy, Porsche and Skull models at a scene resolution of $4096^3$ voxels for each of the l layers

| | **Model** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Lucy $4096^3$** | | | **Porsche $4096^3$** | | | **Skull $4096^3$** | | |
| | **Nodes** | | **Ratio** | **Nodes** | | **Ratio** | **Nodes** | | **Ratio** |
| **l** | **SVO** | **SVDAG** | | **SVO** | **SVDAG** | | **SVO** | **SVDAG** | |
| **0** | 1 | 1 | 1.0 | 1 | 1 | 1.0 | 1 | 1 | 1.0 |
| **1** | 3 | 3 | 1.0 | 2 | 2 | 1.0 | 8 | 8 | 1.0 |
| **2** | 16 | 16 | 1.0 | 12 | 12 | 1.0 | 47 | 47 | 1.0 |
| **3** | 62 | 62 | 1.0 | 73 | 73 | 1.0 | 235 | 235 | 1.0 |
| **4** | 267 | 267 | 1.0 | 421 | 421 | 1.0 | 1004 | 1004 | 1.0 |
| **5** | 1193 | 1192 | 1.0 | 2379 | 2376 | 1.0 | 4343 | 4337 | 1.0 |
| **6** | 5291 | 5271 | 1.0 | 12062 | 11618 | 1.0 | 18170 | 18107 | 1.0 |
| **7** | 22481 | 21972 | 1.0 | 54203 | 49085 | 1.1 | 74103 | 72769 | 1.0 |
| **8** | 91517 | 84658 | 1.1 | 233040 | 193659 | 1.2 | 298845 | 281076 | 1.1 |
| **9** | 366581 | 289927 | 1.3 | 969113 | 633097 | 1.5 | 1192044 | 951264 | 1.3 |
| **10** | 1453101 | 245733 | 5.9 | 3938351 | 528437 | 7.5 | 4688083 | 774408 | 6.1 |
| **11** | 5685858 | 219 | 25962.8 | 15539540 | 255 | 60939.4 | 17958714 | 253 | 70983.1 |

SVOs may contain identical subtrees. These can be retrieved and represented economically by representing such common subtrees in the data structure in only a single fully expanded instance, multi-referenced from the parent nodes. This approach is termed Common Subtrees Merge (CSM). Consequently, the tree transforms into a Directed Acyclic Graph (DAG) and the data structure is designated Sparse Voxel Directed Acyclic Graph (SVDAG). As demonstrated in Figure 1c, showing an example of the encoding, one of the leaf nodes has been identified as such, which can be referenced from the parent node multiple times (in this case twice). The utilisation of CSM resulted in the conservation of space within the leaf node layer. CSM facilitates the reduction of number of nodes located in the respective SVDAG layers, which is a substantial source of compression in comparison to SVO. The probability of finding two or more identical subtrees in the HDS increases with the depth at which their root nodes are located. The maximum level of compression is achieved at leaf nodes, where the total number of nodes is reduced to at most 255. The CHNM of a leaf node is composed of 8 HTs, each with a length of 1b. Consequently, the total length of the CHNM is 8b. This leads us to having only 255 different versions and, hence, different leaf nodes (a leaf node with a $(00000000)_2$ configuration does not exist, as in this case it is a passive octant with no separate node being created for it). The reduction of the number of nodes of each HDS layer when converting an SVO to an SVDAG is illustrated in Table 1.

# 4 SVDAG Parameter Optimization Possibilities

This section addresses the optimization of SVDAG parameters including lengths of the CHNM, the number of different types of child node pointers and their lengths.

## 4.1 Child Node Mask

Section 3 described the construction of SVOs and SVDAGs, where the CHNM of a node is composed of 8 HTs, each with a length of 1b (Figure 2a). When set to *0*, HT represents a passive suboctant in the CHNM that is not subject to further decomposition. Consequently, there is no child node associated with it and no pointer to it in the parent node's pointer array (denoted as × in Figure 2a). If HT is set to *1*, it represents the active suboctant. As a consequence, a child node and a corresponding pointer ($PT_1$) in the parent node's pointer array will exist. The advantage of this construction is the compactness of the CHNM, having a mere 8b. It is also possible to construct the CHNM in such a way that each HT is composed of 2b (Figure 2b). In that case, four distinct HT codes emerge: *00*, *01*, *10* and *11*. The code *00* is employed to flag the existence of a passive suboctant (denoted as × in Figure 2b). The remaining three codes can be used to signal the existence of an active suboctant. These three codes can be used to distinguish which of the three different types of pointers – $PT_1$, $PT_2$ and $PT_3$ – is used. The utilisation of multiple pointer types enables the optimisation of their use when referring to child nodes, thereby facilitating data compression. However, it should be noted that the disadvantage of such CHNM encoding is the increase in the length of its binary code to 16b. The formula for calculating the size of the binary representation of the internal node length $INODE_{len}$ is as follows:

$$INODE_{len} = r + 8 \times ht_{len} + \sum_{x=1}^{3} n_x \times pt_{x\,len} \text{ [b]} \tag{3}$$

Where,

| | |
|---|---|
| $r$ | is the number of bits allocated for CHNM alignment; |
| $ht_{len}$ | is the number of bits constituting the Header Tag in the CHNM ($ht_{len} = 2$); |
| $n_x$ | is the number of active child nodes, to which the $PT_x$ pointers point to; |
| $pt_{x_{len}}$ | is the number of bits forming a $PT_x$ pointer. |

Leaf nodes do not contain a child node pointer array. For the size of their binary representation, denoted $LNODE_{len}$ the following formula applies:

$$LNODE_{len} = r + 8 \times ht_{len} \text{ [b]} \tag{4}$$

Where,

| | |
|---|---|
| $r$ | is the number of bits allocated for CHNM alignment; |
| $ht_{len}$ | is the number of bits constituting the Header Tag in the CHNM ($ht_{len} = 1$); |

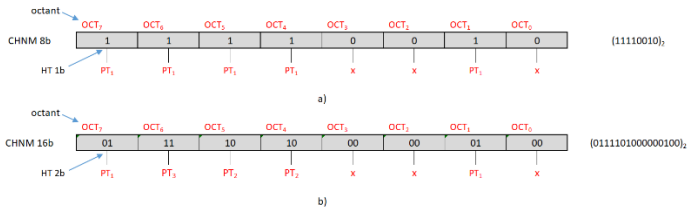


Figure 2

The CHNM comprises HTs for 8 octants ($OCT_0$ to $OCT_7$), a) an 8b CHNM with 1b HTs allows only one type of pointers ($PT_1$); b) a 16b CHNM with 2b HTs allows three types of pointers ($PT_1$, $PT_2$ and $PT_3$)

## 4.2 Addressing

The addresses of individual nodes can be determined within the data structure in a single global address space, as demonstrated in the example in Figures 1b and 1c (for simplicity, addresses are represented as decimal values, in red). Once the root node address has been determined, the allocation of node addresses in the next layer continue. An alternative possibility, illustrated by the example in Figure 1d, is that the global address space is composed of the local address spaces of the individual layers of nodes, where addressing in each layer always restarts from 0. In this scenario, the address within a layer can alternatively be interpreted as the offset from the address where the encoding of the corresponding node layer commences in the global address space of the data structure. To illustrate this, consider a case where pointers permit for a 4GB address space range. In this case, the entire data structure would be subject to this limit, if the global address space was employed. If the pointer is utilised as an offset within the local layer address space, this limit is applied to each of the data structure layer separately. It is not necessary for the entire node to be contained within this address space; it is sufficient for its starting address to be within the address space. The determining factor in the feasibility of encoding a scene into a given HDS is whether the addresses of all nodes in the HDS can be accommodated within the global address space, if a global address space is employed. If the case when local address spaces of individual node layers are employed, the determining factor is the capacity of the local address space to accommodate the addresses of all nodes in the largest node layer.

Rational lengths of pointers to child nodes are 8b, 16b and 32b. A short, 8b pointer includes $2^8$ different addresses. A medium-length pointer of 16b includes $2^{16}$ addresses, while a long pointer of 32b includes up to $2^{32}$ addresses. In the context of the CHNM, the utilisation of a 1b HT results in the selection of only one of these pointer lengths. However, if a 2b HT is employed, it becomes feasible to select a combination of the lengths of the three pointers types. Addressing can be done using words of different lengths. Rational lengths are 8b, 16b and 32b. The capacity of the space that can be addressed is delineated in Table 2. When addressing, there is

a correlation between the pointer and word lengths. For instance, if the lengths of all constituents of nodes, i.e., the length of the CHNM and the length of the child node pointer, are both 32b (as is the case for the SVDAG introduced in [17]), it is rational to use a word length of 32b for addressing. Conversely, e.g., if the objective is to utilise short, 8b pointers, it makes sense to use 8b words for addressing.

Table 2

Size of the space addressable by pointers with alternate lengths 8b, 16b, and 32b when using addressing words of alternate lengths (8b, 16b, and 32b)

| **Pointer [b]** | | **8b** | **16b** | **32b** |
|---|---|---|---|---|
| **Addressed word [b]** | **8b** | 256B | 64KB | 4GB |
| | **16b** | 512B | 128KB | 8GB |
| | **32b** | 1024B | 256KB | 16GB |

## 4.3 Combinations of Child Node Pointer Lengths

In the event of employing a 1b HT, it is possible to utilise a single pointer length per child node, either 8b, 16b or 32b. It is important to note that a pointer length of 8b is impractical even for very small 3D scenes. It can be demonstrated that the utilisation of 16b pointers to child nodes in SVDAGs facilitates the representation of smaller scenes (In order to use 16b words in addressing at the same time, the 8b CHNM needs to be aligned with 8 reserved bits to 16b). Utilising 32b pointers enables the use of a substantial pointer address space, and when aligning the CHNM to 32b, a 32b word can be employed to achieve an address space of up to 16 GB. This approach facilitates the encoding of the scene geometry with a high degree of detail. Conversely, aligning a CHNM to 32b necessitates an augmentation of its binary representation by a factor of 4. Significant portion of the 32b address space remains unused often.

16b CHNM can be employed for frugal data representation, thus allowing the use of up to three types of pointers. To illustrate this point, we may consider a combination of the pointer types $PT_1$ (8b), $PT_2$ (16b) and $PT_3$ (32b). In the case where the length of the addressed word is 8b, if the node address falls within the range $< 0; 255 >$ B, the short 8b pointer $PT_1$ can be used; if it falls within the range $< 0; 65{,}535 >$ B, the use of the medium length 16b pointer $PT_2$ is advisable; and finally, if it falls within the range $< 0; 4{,}294{,}967{,}295 >$ B, the long, 32b pointer $PT_3$ can be used. In the event of having a different combination of pointer types, for instance a 16b $PT_1$, a 16b $PT_2$ and a 32b $PT_3$, at 16b word length for addressing, the first, medium-length 16b pointer ($PT_1$) can be employed to handle the address range $< 0; 131{,}071 >$ B, which corresponds to the first 128KB of the address space. The subsequent medium-length 16b pointer ($PT_2$), can equally address the 128KB range. It is not logical to address the same 128KB as in the case of $PT_1$, therefore resulting in a shift in the $PT_2$ pointer's address range, to the interval $< 131{,}072; 262{,}143 >$ B, thereby addressing 128KB of the address space subsequent

to the address space of the $PT_1$ type pointers. The third, 32b pointer ($PT_3$) is capable of addressing the range $< 0; 8{,}589{,}934{,}591 >$ i.e., 8GB of address space.

## 4.4 Frequency Based Compaction

The aforementioned CSM method allows a particular HDS node to be multi-referenced from a higher layer of nodes. In scenarios involving the use of varying lengths for child node pointers, there exists an opportunity to introduce an optimization that utilizes the address space of shorter child node pointers to store more frequently referenced nodes and to keep the address space reachable exclusively through pointers with longer binary representations for less frequently referenced nodes. This optimization is referred to as Frequency Based Compaction (FBC). The implementation of this optimisation entails sorting the nodes in descending order of their referencing frequency, followed by the assignment of addresses in the address space in ascending order. An illustration of this implementation is provided in Figure 1d, where the referencing frequency of leaf nodes from the root node is arranged in descending order (it should be noted though that the length of all pointers is uniform in this example, thereby rendering the application of FBC meaningless). The addressing frequency of the first 100 most frequently addressed nodes in the 10th layer of SVDAG nodes for the voxelized Angel Lucy model at resolution of $4096^3$ is illustrated in Figure 3. When employing FBC, it is advantageous to avoid using a global addressing space where addresses are assigned continuously across all HDS layers. Rather, separate addressing should be used for each HDS layer. The rationale behind this approach is that, in the event of the former, a single short or medium pointer address space is created for the entirety of the HDS. Utilising offsets from the beginning of the respective node layer ensures the creation of these addressing spaces for short and medium pointers separately in each node layer of the HDS.

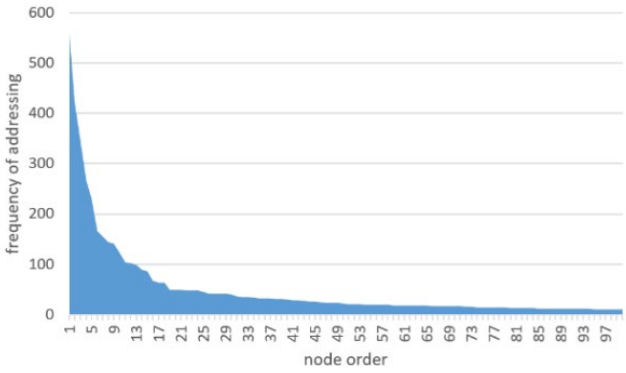


Figure 3
Frequency of addressing of the first 100 most frequently addressed nodes in layer 10 of the SVDAG used to store the Angel Lucy model voxelized at a $4096^3$ resolution

# 5 Test Results and Discussion

This section of the paper summarises test results. Subsection 5.1 presents the voxelized scenes and parameter settings of the tested SVDAG versions. Subsection 5.2 summarizes the results obtained when encoding the scene geometry into the respective HDS versions. Finally, subsection 5.3 discusses the results obtained.

## 5.1 Datasets

Three polygonal surface models, composed of triangle meshes – Angel Lucy ($489 \times 10^3$ triangles), Porsche ($22 \times 10^3$ triangles) and Skull ($80 \times 10^3$ triangles) – were utilised to assess various SVDAG parameter configurations. These models were voxelized to six resolutions, ranging from $128^3$ to $4096^3$, yielding a total of 18 test scenes. The proportion of active voxels to their total number ranged from 3.53% (Skull $128^3$) to 0.03% (Angel Lucy $4096^3$). See Figure 4 for visualizations of the models voxelized at a $512^3$ resolution.

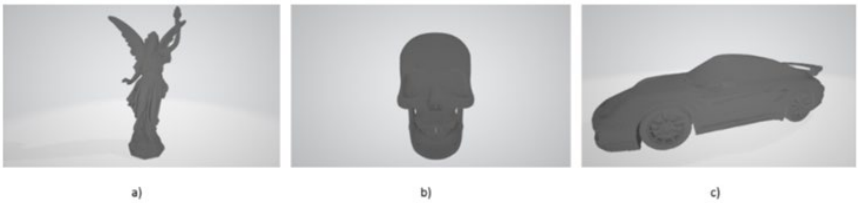


Figure 4
Visualizations of the models to scenes at a $512^3$ voxel resolution:
a) Angel Lucy, b) Skull and c) Porsche

Subsequently, the geometry of the resulting scenes was stored in SVOs and SVDAGs with different parameter configurations. Both the internal and leaf nodes of the SVOs incorporated 8b CHNMs, aligned using reserved bits to 32b. Pointers to child nodes were of only a single type, having a 32b length and the addressing word was also 32b long. The SVDAG denoted as $SVDAG_1$ herein, had identical parameters – its parameters were set as given in [17], where this HDS was introduced. In contrast, the remaining five SVDAG versions ($SVDAG_2$ to $SVDAG_6$) featured 16b CHNMs, permitting three distinct child node pointer types and varying addressing word lengths. A comprehensive overview of the parameter settings is provided in Table 3.

Table 3
Parameter settings for the tested hierarchical data structures

| | **CHNM** | **$PT_1$** | **$PT_2$** | **$PT_3$** | **addr. word** |
|---|---|---|---|---|---|
| **SVO** | 8b (32b*) | 32b | --- | --- | 32b |
| **$SVDAG_1$** | 8b (32b*) | 32b | --- | --- | 32b |
| **$SVDAG_2$** | 16b | 8b | 8b | 32b | 8b |

| | | | | | |
|---|---|---|---|---|---|
| **SVDAG$_3$** | 16b | 8b | 16b | 32b | 8b |
| **SVDAG$_4$** | 16b | 8b | 32b | 32b | 8b |
| **SVDAG$_5$** | 16b | 16b | 16b | 32b | 16b |
| **SVDAG$_6$** | 16b | 16b | 32b | 32b | 16b |

*alignment of 8b CHNMs to 32b, using 24 reserved bits.

## 5.2 Results

The tests were conducted on a computer system with the following configuration: Win 11 Home, 15.6", 1920x1080 FullHD, IPS, AMD Ryzen 7 7435HS, 3.1 GHz, octacore, nVidia GeForce RTX3060, 12 GB RAM, DDR6, 512 GB SSD.

The objective of tests was to determine the total size of the binary representation of the SVO and SVDAG hierarchical data structures with different parameter configurations – such as the number of types and the lengths of pointers per child nodes – for all 18 test scenes. The results obtained for the different models, scene resolutions and HDSs used are summarised in Table 4, with the figures given in KB.

Table 4

SVO and SVDAG binary representation sizes (in KB) with different parameter configurations for the respective models and scene resolutions (best achieved results are shown in grey)

| Size [KB] | **Angel Lucy** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **SVO** | **SVDAG$_1$** | **SVDAG$_2$** | **SVDAG$_3$** | **SVDAG$_4$** | **SVDAG$_5$** | **SVDAG$_6$** |
| **128$^3$** | 53.4 | 31.9 | 12.7 | 10.5 | 13.3 | 15.9 | 15.9 |
| **256$^3$** | 229.0 | 127.0 | 53.3 | 42.8 | 54.5 | 63.5 | 63.5 |
| **512$^3$** | 944.0 | 462.8 | 204.1 | 166.8 | 207.4 | 231.4 | 235.9 |
| **1024$^3$** | 3807.9 | 1523.7 | 724.4 | 623.8 | 737.1 | 793.9 | 829.6 |
| **2048$^3$** | 15160.3 | 4682.5 | 2479.1 | 2155.1 | 2529.1 | 2608.5 | 2708.2 |
| **4096$^3$** | 59581.0 | 14928.3 | 8731.5 | 7637.6 | 8934.0 | 8864.9 | 9138.0 |
| Size [KB] | **Porsche** | | | | | | |
| | **SVO** | **SVDAG$_1$** | **SVDAG$_2$** | **SVDAG$_3$** | **SVDAG$_4$** | **SVDAG$_5$** | **SVDAG$_6$** |
| **128$^3$** | 116.8 | 56.8 | 22.6 | 18.6 | 23.4 | 28.4 | 28.4 |
| **256$^3$** | 540.3 | 222.4 | 95.8 | 75.6 | 98.6 | 111.2 | 111.2 |
| **512$^3$** | 2360.9 | 788.8 | 363.8 | 302.4 | 372.2 | 394.4 | 411.3 |
| **1024$^3$** | 9932.1 | 2629.2 | 1320.6 | 1131.7 | 1356.2 | 1411.3 | 1462.2 |
| **2048$^3$** | 40700.4 | 8918.9 | 4906.6 | 4245.9 | 5034.6 | 5100.3 | 5244.6 |
| **4096$^3$** | 162103.1 | 32128.0 | 18768.7 | 16510.9 | 19438.7 | 19329.8 | 19765.1 |
| Size [KB] | **Skull** | | | | | | |
| | **SVO** | **SVDAG$_1$** | **SVDAG$_2$** | **SVDAG$_3$** | **SVDAG$_4$** | **SVDAG$_5$** | **SVDAG$_6$** |
| **128$^3$** | 186.0 | 100.4 | 42.7 | 34.2 | 43.6 | 50.2 | 50.2 |
| **256$^3$** | 764.9 | 356.7 | 160.7 | 125.7 | 163.7 | 178.4 | 178.4 |
| **512$^3$** | 3099.7 | 1182.8 | 576.2 | 481.1 | 585.8 | 606.1 | 634.1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **$1024^3$** | 12412.5 | 3728.8 | 2012.3 | 1720.8 | 2051.7 | 2053.2 | 2143.3 |
| **$2048^3$** | 49038.1 | 12275.2 | 7239.2 | 6295.1 | 7417.0 | 7260.4 | 7498.8 |
| **$4096^3$** | 189340.6 | 47615.1 | 28290.6 | 25382.3 | 29387.6 | 29419.1 | 30065.2 |

In its uncompressed form, the voxelized scene geometry can be represented by a 3D grid of 1b/vox values. For a scene with a $128^3$ voxel resolution, the size of such a grid amounts to 2,097,152 voxels, and the size of the binary representation of the geometry is 262,144 B, i.e., 256 KB. For a scene with a resolution of $4096^3$ voxels, the size of the binary representation of this grid is then 8 GB. Table 5 provides an overview of the compression ratio achieved when employing the corresponding HDS in comparison to the 1b/vox grid for the designated model, scene resolution and HDS utilised.

Table 5

The achieved compression ratio in comparison to the 1b/vox grid of values for each model, scene resolution and HDS used with different parameter settings

| | **Angel Lucy** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **SVO** | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDAG_6$** |
| **$128^3$** | 4.8 | 8.0 | 20.2 | 24.3 | 19.3 | 16.1 | 16.1 |
| **$256^3$** | 8.9 | 16.1 | 38.5 | 47.8 | 37.6 | 32.2 | 32.2 |
| **$512^3$** | 17.4 | 35.4 | 80.3 | 98.2 | 79.0 | 70.8 | 69.5 |
| **$1024^3$** | 34.4 | 86.0 | 180.9 | 210.1 | 177.8 | 165.1 | 158.0 |
| **$2048^3$** | 69.2 | 223.9 | 423.0 | 486.6 | 414.6 | 402.0 | 387.2 |
| **$4096^3$** | 140.8 | 561.9 | 960.7 | 1098.3 | 938.9 | 946.3 | 918.0 |
| | **Porsche** | | | | | | |
| | **SVO** | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDAG_6$** |
| **$128^3$** | 2.2 | 4.5 | 11.3 | 13.8 | 11.0 | 9.0 | 9.0 |
| **$256^3$** | 3.8 | 9.2 | 21.4 | 27.1 | 20.8 | 18.4 | 18.4 |
| **$512^3$** | 6.9 | 20.8 | 45.0 | 54.2 | 44.0 | 41.5 | 39.8 |
| **$1024^3$** | 13.2 | 49.9 | 99.3 | 115.8 | 96.6 | 92.9 | 89.6 |
| **$2048^3$** | 25.8 | 117.6 | 213.7 | 247.0 | 208.3 | 205.6 | 199.9 |
| **$4096^3$** | 51.7 | 261.1 | 446.9 | 508.1 | 431.5 | 434.0 | 424.4 |
| | **Skull** | | | | | | |
| | **SVO** | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDADG_6$** |
| **$128^3$** | 1.4 | 2.6 | 6.0 | 7.5 | 5.9 | 5.1 | 5.1 |
| **$256^3$** | 2.7 | 5.7 | 12.7 | 16.3 | 12.5 | 11.5 | 11.5 |
| **$512^3$** | 5.3 | 13.9 | 28.4 | 34.1 | 28.0 | 27.0 | 25.8 |
| **$1024^3$** | 10.6 | 35.2 | 65.1 | 76.2 | 63.9 | 63.8 | 61.2 |
| **$2048^3$** | 21.4 | 85.4 | 144.8 | 166.6 | 141.4 | 144.4 | 139.8 |
| **$4096^3$** | 44.3 | 176.2 | 296.5 | 330.5 | 285.4 | 285.1 | 279.0 |

This paper discusses the optimisation of the SVDAG HDS introduced in [17] with a parameter configuration identical to that of $SVDAG_1$. Consequently, Table 6 provides a comprehensive overview of the enhancement in the success rate of HDS compression for the specific model and scene resolution, as well as the voxelization level, when compared to $SVDAG_1$ (this is why the $SVDAG_1$ column contains only 1.00 values).

Table 6

A comparison of the success rate of scene geometry compression for specific models, scene resolutions and HDSs used, compared to the geometry compression rate of $SVDAG_1$

| | **Angel Lucy** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **SVO** | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDAG_6$** |
| **$128^3$** | 0.60 | 1.00 | 2.52 | 3.02 | 2.40 | 2.00 | 2.00 |
| **$256^3$** | 0.55 | 1.00 | 2.38 | 2.97 | 2.33 | 2.00 | 2.00 |
| **$512^3$** | 0.49 | 1.00 | 2.27 | 2.78 | 2.23 | 2.00 | 1.96 |
| **$1024^3$** | 0.40 | 1.00 | 2.10 | 2.44 | 2.07 | 1.92 | 1.84 |
| **$2048^3$** | 0.31 | 1.00 | 1.89 | 2.17 | 1.85 | 1.80 | 1.73 |
| **$4096^3$** | 0.25 | 1.00 | 1.71 | 1.95 | 1.67 | 1.68 | 1.63 |
| | **Porsche** | | | | | | |
| | **SVO** | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDAG_6$** |
| **$128^3$** | 0.49 | 1.00 | 2.52 | 3.06 | 2.43 | 2.00 | 2.00 |
| **$256^3$** | 0.41 | 1.00 | 2.32 | 2.94 | 2.26 | 2.00 | 2.00 |
| **$512^3$** | 0.33 | 1.00 | 2.17 | 2.61 | 2.12 | 2.00 | 1.92 |
| **$1024^3$** | 0.26 | 1.00 | 1.99 | 2.32 | 1.94 | 1.86 | 1.80 |
| **$2048^3$** | 0.22 | 1.00 | 1.82 | 2.10 | 1.77 | 1.75 | 1.70 |
| **$4096^3$** | 0.20 | 1.00 | 1.71 | 1.95 | 1.65 | 1.66 | 1.63 |
| | **Skull** | | | | | | |
| | **SVO** | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDAG_6$** |
| **$128^3$** | 0.54 | 1.00 | 2.35 | 2.94 | 2.30 | 2.00 | 2.00 |
| **$256^3$** | 0.47 | 1.00 | 2.22 | 2.84 | 2.18 | 2.00 | 2.00 |
| **$512^3$** | 0.38 | 1.00 | 2.05 | 2.46 | 2.02 | 1.95 | 1.87 |
| **$1024^3$** | 0.30 | 1.00 | 1.85 | 2.17 | 1.82 | 1.82 | 1.74 |
| **$2048^3$** | 0.25 | 1.00 | 1.70 | 1.95 | 1.66 | 1.69 | 1.64 |
| **$4096^3$** | 0.25 | 1.00 | 1.68 | 1.88 | 1.62 | 1.62 | 1.58 |

The application of multiple child node pointer types in the context of the structures – $SVDAG_2$ to $SVDAG_6$ – has opened up the possibility of employing FBC, and the use of different child node pointer lengths in this context exerted a significant effect on the size of the binary representation of these HDSs. An analysis of the percentage rate of the individual PTs to the total number of child node pointers in the Angel Lucy model, voxelized to the respective resolutions and HDSs ($SVDAG_2$ to $SVDAG_6$) is summarised in Table 7. To facilitate comprehension, the percentage rates (i.e., in terms of the number of pointers of a given type compared to their total count and in terms of the sum of the sizes of the binary representation of the pointers of a given type, compared to the sum of the sizes of the binary representation of all pointers) are listed separately.

Table 7

Proportions of child node pointer types in terms of their counts and their aggregate binary representation sizes for different scene resolutions based on Angel Lucy model - SVDAG2 to SVDAG6

| **$SVDAG_2$** | | **$128^3$** | **$256^3$** | **$512^3$** | **$1024^3$** | **$2048^3$** | **$4096^3$** |
|---|---|---|---|---|---|---|---|
| | **$PT_1$ [8b]** | 81.17 | 78.46 | 75.15 | 69.35 | 60.34 | 51.13 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **No. [%]** | **$PT_2$ [8b]** | 1.62 | 1.07 | 0.98 | 1.24 | 1.60 | 1.90 |
| | **$PT_3$ [32b]** | 17.21 | 20.47 | 23.87 | 29.41 | 38.06 | 46.97 |
| **Size [%]** | **$PT_1$ [8b]** | 53.54 | 48.62 | 43.80 | 36.85 | 28.17 | 21.22 |
| | **$PT_2$ [8b]** | 1.07 | 0.66 | 0.57 | 0.66 | 0.75 | 0.79 |
| | **$PT_3$ [32b]** | 45.39 | 50.72 | 55.63 | 62.49 | 71.08 | 77.99 |

| **$SVDAG_3$** | | **$128^3$** | **$256^3$** | **$512^3$** | **$1024^3$** | **$2048^3$** | **$4096^3$** |
|---|---|---|---|---|---|---|---|
| **No. [%]** | **$PT_1$ [8b]** | 81.38 | 78.58 | 75.20 | 69.36 | 60.33 | 51.11 |
| | **$PT_2$ [16b]** | 18.62 | 21.42 | 20.88 | 17.72 | 19.05 | 20.53 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 3.92 | 12.92 | 20.62 | 28.36 |
| **Size [%]** | **$PT_1$ [8b]** | 68.61 | 64.72 | 56.69 | 44.33 | 33.35 | 24.86 |
| | **$PT_2$ [16b]** | 31.39 | 35.28 | 31.48 | 22.65 | 21.06 | 19.97 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 11.83 | 33.02 | 45.59 | 55.17 |

| **$SVDAG_4$** | | **$128^3$** | **$256^3$** | **$512^3$** | **$1024^3$** | **$2048^3$** | **$4096^3$** |
|---|---|---|---|---|---|---|---|
| **No. [%]** | **$PT_1$ [8b]** | 80.55 | 78.15 | 75.04 | 69.27 | 60.24 | 50.85 |
| | **$PT_2$ [32b]** | 19.45 | 21.85 | 24.96 | 30.73 | 39.76 | 49.15 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Size [%]** | **$PT_1$ [8b]** | 50.87 | 47.20 | 42.91 | 36.04 | 27.47 | 20.55 |
| | **$PT_2$ [32b]** | 49.13 | 52.80 | 57.09 | 63.96 | 72.53 | 79.45 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

| **$SVDAG_5$** | | **$128^3$** | **$256^3$** | **$512^3$** | **$1024^3$** | **$2048^3$** | **$4096^3$** |
|---|---|---|---|---|---|---|---|
| **No. [%]** | **$PT_1$ [16b]** | 100.00 | 100.00 | 97.67 | 89.53 | 81.32 | 73.06 |
| | **$PT_2$ [16b]** | 0.00 | 0.00 | 2.33 | 5.42 | 4.96 | 4.34 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 0.00 | 5.05 | 13.72 | 22.60 |
| **Size [%]** | **$PT_1$ [16b]** | 100.00 | 100.00 | 97.67 | 85.22 | 71.51 | 59.58 |
| | **$PT_2$ [16b]** | 0.00 | 0.00 | 2.33 | 5.16 | 4.36 | 3.54 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 0.00 | 9.62 | 24.13 | 36.88 |

| **$SVDAG_6$** | | **$128^3$** | **$256^3$** | **$512^3$** | **$1024^3$** | **$2048^3$** | **$4096^3$** |
|---|---|---|---|---|---|---|---|
| **No. [%]** | **$PT_1$ [16b]** | 100.00 | 100.00 | 97.67 | 89.30 | 81.16 | 72.98 |
| | **$PT_2$ [32b]** | 0.00 | 0.00 | 2.33 | 10.70 | 18.84 | 27.02 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| **Size [%]** | **$PT_1$ [16b]** | 100.00 | 100.00 | 95.45 | 80.67 | 68.29 | 57.46 |
| | **$PT_2$ [32b]** | 0.00 | 0.00 | 4.55 | 19.33 | 31.71 | 42.54 |
| | **$PT_3$ [32b]** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

## 5.3 Discussion

The preceding synopsis of the findings of the conducted tests indicated that, in accordance with the prevailing expectations, the utilisation of CSM in SVDAGs led to an enhancement in the compactness of geometry representation, surpassing the performance of SVOs. This was evident even in the $SVDAG_1$ version, employing 32b CHNMs and maintaining a consistent 32b child node pointer length.

In versions $SVDAG_2$ to $SVDAG_6$, 16b CHNMs without reserved bits have been employed. This represents merely half the length of the binary representation of the CHNM, as opposed to its 32b length in the case of SVO and $SVDAG_1$ (8b CHNM + 24b reserved space). This has been identified as a significant source of

compression gain (16b per internal node) with the $SVDAG_2$ to $SVDAG_6$ structures. Concurrently, this facilitates the utilisation of multiple child node pointer types, potentially exhibiting disparate binary representation lengths, and the employment of FBC. One of the pointer types ($PT_3$) was invariably selected to have a length of 32b to allow the encoding of more detailed scenes. The remaining two pointer types, designated $PT_1$ and $PT_2$, were selected as combinations of 8b, 16b, and 32b lengths, respectively, as follows: both 8b in $SVDAG_2$; 8b and 16b in $SVDAG_3$; 8b and 32b in $SVDAG_4$, both 16b in $SVDAG_5$ and 16b and 32b in $SVDAG_6$. It is noteworthy that the combination of 32b and 32b is not utilised, as this would result in the length of the pointers remaining constant at 32b. The shorter overall binary representation of the CHNM-related part of the node, together with the fact that some of the pointers to child nodes have a binary representation size smaller than 32b, allows versions $SVDAG_2$ to $SVDAG_6$ to always outperform the original $SVDAG_1$ in binary representation size. The amount of space saved $S$, when using $SVDAG_2$ to $SVDAG_6$ compared to using $SVDAG_1$, can be calculated using the formula:

$$S = 16 \times INODE_{num} + n \times LNODE_{num} + \sum_{x=1}^{3} (32 - PT_{x_{len}}) \times PT_{x_{num}} \; [b] \tag{5}$$

Where,

| | |
|---|---|
| $INODE_{num}$ | is the number of internal nodes of HDS; |
| $LNODE_{num}$ | is the number of leaf nodes of HDS; |
| n | is 24 for $SVDAG_2$ to $SVDAG_4$ and 16 for $SVDAG_5$ and $SVDAG_6$; |
| $PT_{x_{len}}$ | is the length in bits of the pointer of type $PT_x$; $x \in < 1; 3 >$; |
| $PT_{x_{num}}$ | is the overall number of pointers of type $PT_x$; $x \in < 1; 3 >$ in HDS. |

For the SVO and $SVDAG_1$ structures, the use of 32b addressing words is an advantage, which, together with the use of 32b child node pointer lengths, allows the use of an addressing up to 16GB. For SVDAG-versions utilising 8b pointers, such as $SVDAG_2$ to $SVDAG_4$, an 8b address word length has been utilised. This configuration results in a reduction of the available addressable space to 4 GB (for $SVDAG_4$, the adoption of two 32b pointer types increases the addressable space to 8 GB). For SVDAGs with 16b shortest pointers, i.e., $SVDAG_5$ and $SVDAG_6$, a 16b address word length is employed, resulting in 8 GB and 16 GB of addressable space.

In the transition from SVO to SVDAG, the reduction in the number of leaf nodes has a substantial positive effect also on the overall size of the HDS. Nevertheless, it is important to note that the number of pointers pointing to them from pre-leaf nodes remains significant. Given that the maximum leaf node count in SVDAGs is 255, all of these fit into the 8b pointer address space. This observation favours data structures $SVDAG_2$ to $SVDAG_4$ having exactly this pointer length type and thus capable of addressing all leaf nodes exclusively with 8b child node pointers.

For each model and all SVDAGs with multiple pointer lengths ($SVDAG_2$ to $SVDAG_6$), along with an increase of the resolution to which a given model has been voxelized, the positive effect of using multiple pointer lengths progressively diminishes. As demonstrated in Table 7, this phenomenon can be attributed to the fact that, the more complicated the scene gets, the more the proportion of short (8b) and medium-length (16b) child node pointers declines in favour of long (32b) pointers, both in terms of their number and, to a greater extent, in terms of their total binary representation length.

To determine the size of the $SVDAG_2$ to $SVDAG_6$ data structure prior to its construction, a statistical analysis of the related classical $SVDAG_1$ HDS can be performed. This analysis starts at the leaf node layer, where the node length is adjusted for each leaf node to ensure addressability using the optimal (shortest) pointer type from the optimized SVDAG. The process is then applied recursively from the leaf layer up to the root layer, when for each pair of adjacent node layers $l$ and $l-1$, the following steps are performed:

**Step 1** Sort the nodes in layer $l$ based on the frequency of their referencing from layer $l-1$ (to enable Frequency-Based Compaction)

**Step 2** Assign the nodes of layer $l$ (and thus the pointers to them) to the addressing space of specific pointer types

**Step 3** Update the pointer length information for each pointer in the parent layer $l-1$, according to the assignment of those pointers to specific pointer types

**Step 4** Assign updated information about binary representation length of the nodes in layer $l-1$ (due to changes in CHNM size and the pointer lengths to child nodes)

**Step 5** Determine the total size of binary representation of layer $l-1$

By iteratively executing these steps, it is possible to calculate the binary sizes of all node layers in the potentially optimized SVDAG, and therefore, the total size of its binary representation. Applying this statistical analysis to various configurations of the optimized SVDAGs (i.e., different combinations of pointer types) allows us to identify the optimal configuration of pointer types that minimizes the overall size of the binary representation of optimised SVDAG.

In the tests, the most compact HDS was achieved for all models and resolutions by combining 8b $PT_1$, 16b $PT_2$, and 32b $PT_3$ pointer lengths with an 8b address word length, i.e., in $SVDAG_3$.

The traversal time of the optimized SVDAG data structures ($SVDAG_2$ to $SVDAG_6$) is influenced, in comparison to the original $SVDAG_1$, by several negative factors. While $SVDAG_1$ is 32b-aligned, which is GPU-friendly, $SVDAG_2$ to $SVDAG_4$ are only 8b-aligned, and $SVDAG_5$ and $SVDAG_6$ are 16b-aligned. Addressing using 16b words within $SVDAG_5$ and $SVDAG_6$ requires additional decoding operation.

$SVDAG_2$, $SVDAG_4$, $SVDAG_5$ and $SVDAG_6$ versions implement two pointer types of the same length and it requires additional operation of shifting addressing space for one of them. The introduction of three different pointer types also complicates CHNM processing, determining the offset of the child node pointer within the parent node and the actual loading of this pointer. On the other hand, the optimized SVDAG versions are more compact, which helps reduce potential cache misses. All those factors result into slower traversal in the optimized versions of SVDAG compared to the original $SVDAG_1$.

Table 8 shows traversal time in µs for individual models voxelized to scenes with resolution of $4096^3$ voxels. Traversal was performed in parallel from the root node to the leaf nodes for all active voxels in the scene using nVidia RTX 3060 graphics card. Table 9 and Figure 5 show the ratio between the traversal times of $SVDAG_1$ and each optimized SVDAG variant. Results show that traversal speed was reduced by 15% to 25% when optimized versions of SVDAG were used.

Table 8

SVDAG traversing time in µs for all active voxels in the scene with resolution of $4096^3$ vox

| time [µs] | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDAG_6$** |
|---|---|---|---|---|---|---|
| **Angel Lucy $4096^3$** | 1839.1 | 2169.9 | 2276.4 | 2248.7 | 2311.9 | 2284.5 |
| **Porsche $4096^3$** | 4397.1 | 5358.0 | 5679.1 | 5598.2 | 5825.4 | 5784.6 |
| **Skull $4096^3$** | 5299.2 | 6306.8 | 6503.4 | 6585.3 | 6617.8 | 6557.7 |

Table 9

Ratio between SVDAG1 and SVDAGn traversing time in scene with resolution of $4096^3$ vox

| Ratio | **$SVDAG_1$** | **$SVDAG_2$** | **$SVDAG_3$** | **$SVDAG_4$** | **$SVDAG_5$** | **$SVDAG_6$** |
|---|---|---|---|---|---|---|
| **Angel Lucy $4096^3$** | 1.00 | 0.85 | 0.81 | 0.82 | 0.80 | 0.81 |
| **Porsche $4096^3$** | 1.00 | 0.82 | 0.77 | 0.79 | 0.75 | 0.76 |
| **Skull $4096^3$** | 1.00 | 0.84 | 0.81 | 0.80 | 0.80 | 0.81 |

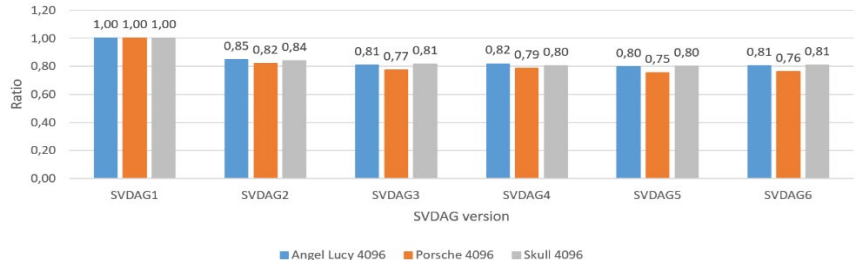


Figure 5

Ratio between SVDAG1 and SVDAGn traversing time in scene with resolution of $4096^3$ vox

## Conclusions

This paper addressed the problem of representing the geometry of voxelized 3D scenes, through hierarchical data structures, with a focus on SVDAGs. The objective was to analyse the potential of parameter optimizations of this data structure, including the CHNM size and child node pointer types and lengths.

The analysis of these possibilities was complemented by empirical testing, through experiments performed on encoding the geometry of polygonal surface models, voxelized at different resolutions.

The findings indicate that the employment of an SVDAGs using 16b CHNM and three types of pointers to child nodes is able to lower the binary size of SVDAG HDS and in tests was empirically determined that using three child node pointer types of lengths 8b, 16b, and 32b, respectively, emerged as the optimal configuration of child node pointer types. This is evidenced by a substantial enhancement in data compression, ranging from 1.88 (at a scene resolution of $4096^3$ voxels) to 3.06 (at a scene resolution of $128^3$ voxels), when compared with conventional SVDAG HDS.

Concurrently, the positive impact of employing a greater variety of child node pointer types was demonstrated for all settings of combinations of their lengths; however, it was also observed that the effect diminished with increasing scene detail for all pointer combinations. Traversal speed, when optimized SVDAGs were used, was reduced by 15% to 25%, in comparison to traversal speed of a conventional SVDAG.

**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] Klinger, A.; Dyer, C.R.: Experiments in Picture Representation Using Regular Decomposition. Computer Graphics and Image Processing, Vol. 5, 1976, No. 1, pp. 68-105, doi: 10.1016/S0146-664X(76)80006-8

[2] Gargantini, I.: An Effective Way to Represent Quadtrees. Communications of the ACM, Vol. 25, 1982, No. 12, pp. 905-910, doi: 10.1145/358728.358741

[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] Meagher, D. J. R.: Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3-D Objects by Computer. Technical Report No. IPL-TR-80-111, Rensselaer Polytechnic Institute, Troy, NY, 1980

[6] Meagher, D. J. R.: Geometric Modeling Using Octree Encoding. Computer Graphics and Image Processing, Vol. 19, 1982, No. 2, pp. 129-147, doi: 10.1016/0146-664X(82)90104-6

[7] Meagher, D. J. R.: The Octree Encoding Method for Efficient Solid Modeling. Technical Report IPL-TR-032, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, New York, 1982

[8] 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

[9] 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

[10] 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/10.1145/2601097.260122

[11] 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 February-1 March 2015; pp. 25-30, ISBN: https://doi.org/10.1145/ 2699276.2699284

[12] Kämpe, V.; Rasmuson, S.; Billeter, M.; Sintorn, E.; Assarsson, U. Exploiting Coherence in Time-Varying Voxel Data. In Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, Redmond, WA, USA, 27-28 February 2016; pp. 15-21, ISBN 9781450340434, https://doi.org/10.1145/2856400.2856413

[13] Baert, J.; Lagae, A.; Dutré, Ph. Out-of-Core Construction of Sparse Voxel Octrees, Computer Graphics Forum 2014, 33, pp. 220-227, ISSN 0167-7055, https://doi.org/10.1111/cgf.12345

[14] 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

[15] 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

[16] 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

[17] 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

[18] Villanueva, A. J.; Marton, F.; Gobetti, E. Symmetry-aware Sparse Voxel DAGs. In Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '16), Redmond, WA, USA, 27-28 February, 2016; pp. 7-14, ISBN https://doi.org/10. 1145/2856400.2856420

[19] Vokorokos, L.; Madoš, B.; Bilanová, Z. PSVDAG: Compact Voxelized Representation of 3D Scenes Using Pointerless Sparse Voxel Directed Acyclic Graphs. CAI, 2020, 39, pp. 587-616, ISSN 1335-9150 (print); 2585-8807 (online) https://doi.org/10.31577/cai_2020 _3_587

[20] 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

[21] van der Laan, R.; Scandolo, L.; Eisemann, E. Lossy Geometry Compression for High Resolution Voxel Scenes. Proceedings of the ACM on Computer Graphics and Interactive Techniques 2020, 3, pp. 13, EISSN: 2577-6193, https://doi.org/10.1145/3384541

[22] 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