Computing and Informatics, Vol. 39, 2020, 587–616, doi: 10.31577/cai_2020_3_587

PSVDAG: COMPACT VOXELIZED REPRESENTATION OF 3D SCENES USING POINTERLESS SPARSE VOXEL DIRECTED ACYCLIC GRAPHS

Liberios Vokorokos, Branislav Madoš, Zuzana Bilanová

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
e-mail: {liberios.vokorokos, branislav.mados, zuzana.bilanova}@tuke.sk

Abstract. This paper deals with the issue of geometry representation of voxelized three-dimensional scenes using hierarchical data structures. These include pointerless Sparse Voxel Octrees that have no pointers on child nodes and allow a compact binary representation. However, if necessary, there is a possibility to reconstruct these pointers for rapid traversing. Sparse Voxel Directed Acyclic Graphs added 32-bit pointers to child nodes and merging of common subtrees, which can be considered lossless compression. By merging common subtrees, no decompression overhead occurs at the time of traversing. The hierarchical data structure proposed herein – the Pointerless Sparse Voxel Directed Acyclic Graph – incorporates the benefits of both – pointerless Sparse Voxel Octrees (by avoiding storing pointers on child nodes) and Sparse Voxel Directed Acyclic Graphs (by allowing the merging of common subtrees due the introduction of labels and callers). The proposed data structure supports the quick and easy reconstruction of pointers by introducing the Active Child Node Count. It also potentially allows Child Node Mask compression of its nodes. This paper presents the proposed data structure and its binary-level encoding in detail. It compares the effectiveness of the representation of voxelized three-dimensional scenes (originally represented in OBJ format) in the proposed data structure with the data structures mentioned above. It also summarizes statistical data providing a more detailed description of the various parameters of the data structure for different scenes stored in multiple resolutions.

Keywords: Lossless data compression, sparse voxel octrees, SVO, sparse voxel directed acyclic graph, SVDAG, pointerless sparse voxel directed acyclic graph, PSVDAG, symmetry-aware sparse voxel directed acyclic graph, SSVDAG

Mathematics Subject Classification 2010: 68-P30

1 INTRODUCTION

The initial motivation for volumetric data representation was the need to process and visualize three-dimensional scientific, medical, and industrial data. Volume datasets are mostly spatially uniform, regular grids of scalar or vector values. They may be obtained through the acquisition using specialized appliances that fit for this purpose or represent the results of simulations or calculations. Computed Tomography (CT), MicroCT, or Magnetic Resonance Imaging (MRI) are devices suitable for data acquisition in medical applications. As far as the industries are concerned, Industrial Computed Tomography can be a source of data for assembly inspection and measurements, flaw detection, failure analysis, or reverse engineering. Voxelized polygonal three-dimensional models may act as another, often used, source of volume data.

As the development of volumetric data visualization continued (including the development of hardware to support it), its employment has also shifted to computer graphics, visual arts, and computer-based entertainment, including computer games, augmented or virtual reality. Volume rendering is also widely used in the movie industry. Notable examples of volumetric graphic visualization include the motion pictures XXX, Lord of the Rings, The Day After Tomorrow, Pirates of the Caribbean, or The Mummy [1]. The advantages of voxelized graphics include the possibility of implementing effects such as smoke, fog, snow, fire, and more.

Complex voxelized three-dimensional scenes, even taking only the geometry of the scene into account, can represent a significant amount of data. An example may be the geometry of a scene having a resolution of $64\,K^3$ ($65\,536 \times 65\,536 \times 65\,536$) voxels. Using a regular three-dimensional grid of 1 b per voxel for its representation, up to 32 TB of space is required to store it in an uncompressed form. Processing this type of data in its uncompressed form may be inefficient. If the goal is the real-time or interactive processing, requiring the storage of the entire amount of data in the computer's memory or the graphics card, this task may not be feasible. Especially when reasonable hardware resources and off-the-shelf hardware solutions are required. That is why significant effort is devoted to developing more compact representations of voxelized three-dimensional data (with lossless compression).

In this context, various data structures have emerged over the decades of development. Hierarchical data structures based on octant trees have gained considerable popularity. Among other advantages, they allow the efficient work with empty subspaces or an implicit Level of Detail (LOD) mechanism. A more comprehensive overview of GPU-friendly volume data representation and rendering is available in [2].

Sparse Voxel Octrees (SVO) are used as a hierarchical data structure to represent the geometry of voxelized three-dimensional scenes. SVO allows carving out empty suboctants, i.e., those containing no active voxels.

In 2013, with the emergence of Sparse Voxel Directed Acyclic Graphs (SVDAG), GPU friendly directed acyclic graphs with Common Subtree Merging (CSM) were

introduced. In this concept, a fully developed instance of the particular subtree, referenced (multiple times) by child node pointers, replaces two or more identical subtrees. CSM can be considered a lossless compression of the hierarchical data structure. In 2016, a GPU friendly Symmetry-aware Sparse Voxel Directed Acyclic Graphs (SSVDAG) emerged. SSVDAG added the CSM option, provided that transformations of these subtrees – namely reflective symmetry transformation – are used. SSVDAG allows further modification of the data structure – clustering leaf nodes, to form 4^3 grids of voxels – for further increase of the compression ratio.

On the other hand, in this context, it is necessary to observe how this data compaction manifests itself in the use of the data structure. The decompression overhead must be as small as possible. The reason is the need for on-the-fly decompression during traversing. This issue may also be considered a trade-off between the size of the binary representation of the data and the time it takes to perform operations. Data optimization performs data compaction while preserves the previous service functionality [3].

Modifying the presence, structure, and length of the binary representation of pointers to the child nodes is another factor that can reduce the size of the data structures mentioned above. An example is the SSVDAG data structure having 0, 16, and 32-bit pointers, which is a refinement over the previous SVDAG data structure, only having 0 and 32-bit pointers. On the other hand, pointerless SVOs have no pointers at all.

In this work, the authors attempted to create a hierarchical data structure without any pointers on child nodes. Therefore it will be significantly more compact than SVDAG and, considering the way of encoding information about the decomposition of individual suboctants, approaching pointerless SVO. On the other hand, like SVDAG, it will enable compaction of the data structure via the CSM, even without the presence of child node pointers, using labels and callers with a compact variable length of their binary representation, introduced in this paper. Frequency-based compaction will promote further compacting of labels and callers. Additional information facilitates the reconstruction of child node pointers into a form allowing quick and efficient traversal.

In more modern hierarchical data structures, the number of mask bits of potential child nodes increases from 1 b to 2 b, increasing the length of the Child Node Mask from 8 b to 16 b. Therefore a further goal was to investigate the possibility of compacting the Child Node Mask (CHNM) of the respective nodes. Pointers optimization offers the potential to rationalize the binary representation of this part of the hierarchical data structure node.

The contribution hereof lies in the following:

- the design of a Pointerless Sparse Voxel Directed Acyclic Graph (PSVDAG) hierarchical data structure allowing a compact pointerless representation of the geometry of three-dimensional scenes;
- the implementation of common subtrees merge with the absence of child node pointers at the same time. Introduction of the concept of labels and callers that

are replacing particular pointers of the hierarchical data structure, is enabling the CSM;

- the length optimization of the binary representation of labels and callers, using variable-length and frequency-based compaction;
- the introduction of Active Child Node Count information to facilitate and accelerate pointers reconstruction;
- the introduction of a variable-length representation of Child Node Mask, with potential lossless compression.

The structure of the paper is as follows:

Section 2 of the paper deals with the linearization of multi-dimensional data using space-filling curves and hierarchical data structures designed for compact representation using lossless compression. Due to the vast number of papers published in this field, this section contains only a selection of works related to the work presented herein.

Section 3 hereof contains an overview of the features, advantages, disadvantages, and explanation of binary-level representations of pointerless SVO and SVDAG. Their favorable properties also represent the benefits of the work described herein.

Section 4 introduces the proposed Pointerless Sparse Voxel Directed Acyclic Graph (PSVDAG) hierarchical data structure, along with a detailed description of its properties, including the concepts of Labels/Callers, variable child node mask length and active child node count, introduced in this work. There is also a detailed description of its binary-level representation.

Section 5 of the paper summarizes the results of tests performed on voxelized 3D scenes with the geometry stored as pointerless SVO, SVDAG, and PSVDAG. Section intends to test the storage efficiency of the data structure proposed in this paper. The examination of the influence of the use of z-order and Hilbert space-filling curves takes part in this section of the paper. It also summarizes sources of compression gains of PSVDAG, compared to SVO and SVDAG.

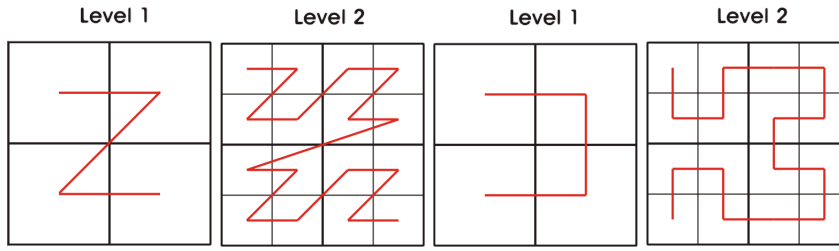
Section 6 summarizes the conclusions drawn from the evaluation of the test results, which details the previous section of this paper.

2 RELATED WORKS

Due to the vast number of papers published in the field of space-filling curves and hierarchical data structures designed to represent multi-dimensional data, this section contains only a selection of publications closely related to the solution proposed and presented herein.

Linearization of multi-dimensional grids of data. Peano and Hilbert presented Space-Filling Curves (SFC) at the end of the 19^{th} century, later followed

by Moore, Lebesgue, Sierpinski, and others [4, 5]. A very popular is the Hilbert Space-Filling Curve (HSFC), introduced by David Hilbert in 1891 [6]. Computer science often uses the Hilbert curve because of its better locality-preserving behavior. The Morton order introduced in 1966 [7], referenced as the Morton sequence, the Morton code, or z-order curve, is often used for the linearized representation of n-dimensional grids of data in computer graphics. As an example of the usage of the z-order curve and Hilbert curve for the filling of two-dimensional space to the different levels, see Figure 1.



a) To the first level b) To the second level c) To the first level d) To the second level

Figure 1. Linearization of two-dimensional space using space-filling z-order curve and alternatively using Hilbert curve

Hierarchical representation of two dimensional data. The utilization of quadtrees to represent images as trees is described in [8, 9, 10, 11]. A discussion on the possibility of using quadtrees for an efficient representation of two-dimensional data using the technique of Common Subtree Merging (CSM) is in [12]. In addition to the empty subsquares uniformly filled with passive pixels, the referenced work also describes the use of subsquares homogeneously filled with active pixels. An appropriate data structure, the Common Subtree Merge Quadtree (CSM-Quadtree), has been created.

The two-dimensional template-based encoding (2DTE) technique, mentioned in [13], was meant for the construction of quadtrees for the use in the cartography field. It represents the lossless compression based on the merging of common subtrees with the use of template mapping and the Morton sequence. This approach extended to three-dimensional voxel data is mentioned in [14].

Hierarchical representation of three-dimensional data. Octrees – a hierarchical data structure – serve to represent three-dimensional data for decades. Works from the 1980s include Srihari [15], Rubin and Whitted [16], Jackins and Tanimoto [17], Meagher [18, 19, 20]. In [21], the author focused on the use of hierarchical data structures such as the octree to speed up the determination of objects intersected by rays emanating from the viewpoint. An algorithm for raytracing octrees consisting of volumetric data is in [22, 23].

Sparse Voxel Octrees (SVO) is a hierarchical data structure designed to represent three-dimensional scenes using octrees. SVO allows for a frugal representation of empty subspaces – the subtrees associated with these empty subspaces are not represented in the data structure.

The octree space decomposition is part of the compression method presented in [24]. It is specialized for point-sampled models explicitly aimed at densely sampled surface geometry.

In 2010, the Efficient Sparse Voxel Octrees (referenced in sources as ESVO), a hierarchical data structure, was introduced in [25]. As an octree hierarchical data structure, it allows carving out empty spaces. It also removes entire subtrees in case there is a possibility to use contour information. It is a representation with a length of 32 b per node, consisting of a 24 b contour pointer and an 8 b contour mask. This modification allows an increase of the geometric resolution, allowing a more compact representation of smooth surfaces, while accelerating ray casting.

In 2013, the Sparse Voxel Directed Acyclic Graph (SVDAG), a hierarchical data structure, was introduced in [26]. SVDAG transforms sparse voxel octrees into oriented acyclic graphs (DAG). This approach uses 8 b masks of child nodes to represent the decomposition of octants to suboctants. The node representation includes a variable number of 32 b pointers pointing to the locations of the active child nodes. This data structure replaces multiple identical subtrees by a single representation of such a subtree. It allows the application of common subtree merging when several child pointers point to the root node of such a subtree. The structure aligns the child node mask to 32 b with unused 24 b (see Figure 2).

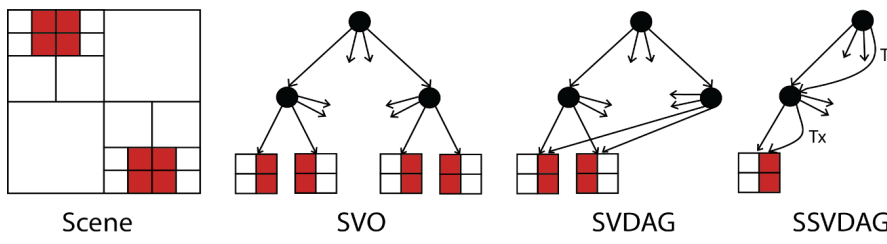


Figure 2. Original scene represented as the 2D grid of pixels (Scene) and as the Sparse Voxel Octree (SVO), Sparse Voxel Directed Acyclic Graph (SVDAG) and Symmetry-aware Sparse Voxel Directed Acyclic Graph (SSVDAG). Tx represents similarity transformation along the axis x, Ty represents similarity transformation along the axis y.

In 2016, an evolution of the SVDAG hierarchical data structure, the Symmetry-aware Sparse Voxel Directed Acyclic Graph (SSVDAG), appeared in [27]. This data structure uses a 16 b child node mask. Each child node describes a 2 b Header Tag (HT) and allows further data compression with a minimal impact

on the decompression overhead. SSVDAG allows merging of common subtrees that are identical when a similarity transformation – a reflection transformation in one or more main axes of a scene – is applied. Another benefit is the use of frequency-based pointer compaction. It orders nodes in each layer of the tree according to their referencing frequency – from the most to the least frequently referenced nodes. The most commonly referenced nodes reside in an area addressed by short, 16 b pointers with a 01 header tag. The less often referenced nodes are in an area addressed by long, 32 b ones with a 10 header tag. Extra-long 33 b pointers have an 11 header tag. A 3 b information – as the part of the vector – is indicating the relevant similarity transformation, as depicted in Figure 3. The last two levels of nodes are forming voxel grids of 4^3 voxels.

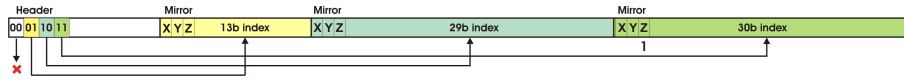


Figure 3. Symmetry-aware Sparse Voxel Directed Acyclic Graph (SSVDAG) binary representation of inner nodes with short, long and extra-long pointers to the child nodes

In 2019, a small modification of the SSVDAG hierarchical data structure, the SSVDAG*, was introduced in [28]. As part of this modification, pointers with the header tag 11, which allowed 33-bit addresses and an indication of reflective transformations – were replaced with 16-bit ones with an identical header tag without transformation indication. Compared to short 16-bit ones of the SSVDAG, these allow an 8-fold increase in address space, provided there is no need for reflective transformations. Thus, some long, 32-bit pointers become 16-bit, with the advantage that these point to more frequently referenced nodes. Compared to SSVDAG, this allows further compression. On the other hand, a reduction of the total addressable space occurs.

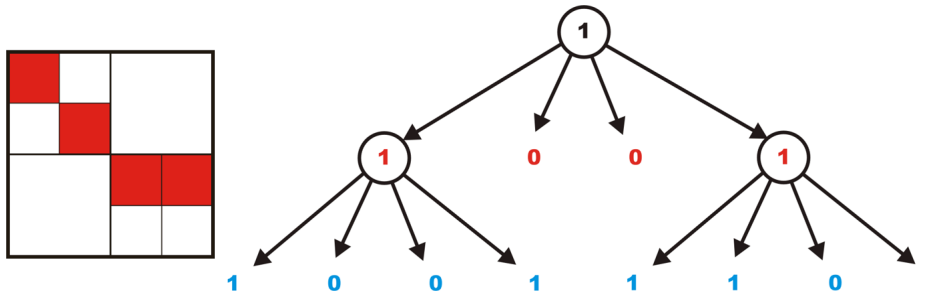
Out-of-core construction algorithms. Out-of-core construction of Sparse Voxel Octrees from triangle meshes was introduced in [29] in 2013. The algorithm consists of two basic steps. The first is a voxelization process. It transforms the triangles representing the scene to the intermediate product – a high-resolution three-dimensional voxel grid. In the second step, the transformation of this intermediate product into Sparse Voxel Octree (SVO) occurs. The algorithm allows the size of the binary representation of the input mesh of triangles, the output SVO, and the intermediate product – a three-dimensional voxel grid generated at a high resolution and represented by a Morton order – to exceed the available computer operating memory by far. Compared to the in-core algorithm, it uses only a fraction of the memory while maintaining the use of comparable time.

3 HIERARCHICAL DATA STRUCTURES

This section presents a brief overview of pointerless Sparse Voxel Octrees (SVO) and Sparse Voxel Directed Acyclic Graphs (SVDAG). It provides information about their basic properties, advantages, disadvantages, and binary-level encoding.

3.1 Pointerless Sparse Voxel Octrees Overview

Sparse Voxel Octrees (SVO) are hierarchical data structures enabling the representation of a voxelized three-dimensional scene with a frugal way of encoding empty subspaces. The SVO root node represents the entire three-dimensional scene. SVO divides this space into eight octants and contains one bit for each of them. This bit is set to 0 if the octant does not comprise any active voxels (i.e., the octant is empty); thus, it will not decompose. This bit is set to 1 if the particular octant contains at least one active voxel. If this active octant consists of more than a single voxel, decomposition to suboctants follows recursively. The pointerless version of SVO does not contain any pointers to child nodes.



a) A two-dimensional pixel grid

b) An SVO in tree form

1(1(1001)001(1100))

c) A binary representation of SVO using parentheses to illustrate the recursive decomposition of active sub-quadrants

11100100111 00

d) The final binary SVO representation

Figure 4. An illustration of the SVO hierarchical data structure to simplify the example, used to encode a two-dimensional pixel grid using a z-order curve for linearization

Figure 4 a) depicts a two-dimensional grid having 4×4 pixels, with the active pixels indicated in red, and the inactive pixels in white. SVO hierarchical data structure, in the form of the tree, is shown in Figure 4 b). There are active quadrants 0 and 3, subsequently decomposed. The binary representation of the pointerless SVO structure – with parentheses added to illustrate the decomposition points – is in Figure 4 c). The final binary encoding of the pointerless SVO data structure for

this pixel grid, is stated in Figure 4 d). In the hierarchical data structure, each decomposition of the tree nodes generated four bits of information concerning the potential child nodes. With three-dimensional voxel grids, there are 8 bits of eight possible child nodes for each octant decomposition of the respective octant. Those 8 bits are forming the Child Node Mask (CHNM).

The advantage of the hierarchical data structure constructed as stated above is its compact binary representation, since each node decomposition generated only eight bits. Pointerless data structure has its advantage in the absence of pointers to child nodes that represent vast amounts of data. In the most unfavorable case, if there are all eight pointers to child nodes, using 32 bits per pointer, an additional 256 b would have to be allocated. The pointerless data structure constructed, as stated above, is suitable for streaming or archiving purposes. The downside is in its cumbersome and time-consuming traversing at the time of scene visualization. Another disadvantage of such a data structure is that it does not allow the compact representation using CSM.

3.2 Sparse Voxel Directed Acyclic Graph Overview

The Sparse Voxel Directed Acyclic Graph (SVDAG) is a hierarchical data structure in the form of a directed acyclic graph. It consists of nodes containing child nodes masks of these nodes and the corresponding pointers to child nodes. The node mask is composed of 8 bits, assigned to the respective suboctants. A bit set to 0 represents an empty suboctant, containing no active voxel. In this case, in the data structure, the corresponding child node – including the child node pointer – is not present. A bit set to 1 represents an active suboctant, containing at least one active voxel. For such a suboctant, a child node is created in the data structure, referenced by the corresponding node pointer. The concatenation of 24 unused bits aligns the mask to 32 bits.

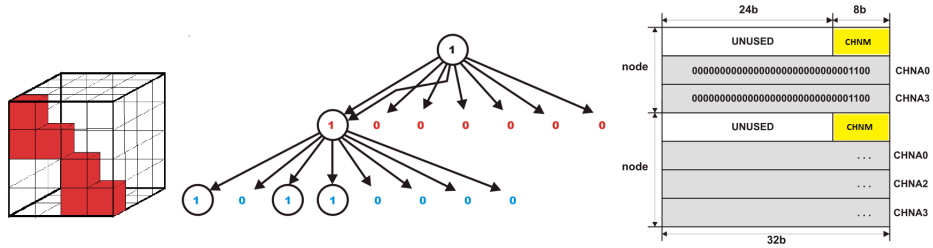
A block of pointers to the existing child nodes follows the node mask. Their count is from the range $\langle 1; 8 \rangle$. Each one has 32 bits and points to a memory location where a binary representation of the corresponding child node is stored. All node parts and thus, each node and the whole data structure are aligned to 32 bits, as shown in Figure 5 c). Multiple pointers of different but also the same node may point to the same memory location. Therefore multiple identical subDAGs may be represented by fully encoding only a single copy of subDAG, with several references to it by child node pointers. SVDAG thus enables CSM.

The length of the SVDAG node binary-level representation l having n pointers to child nodes can be calculated as

$$l = (n + 1) \times 32 \text{ [b]}. \tag{1}$$

Compared to pointerless SVOs, the advantage of the SVDAG data structure lies in the introduction of pointers to child nodes, which allows fast graph traversal when processing and visualizing the scene. Another advantage is the possibility of merging

common subtrees, through multiple references to a particular node by several child node pointers. It allows significant compaction of the tree's binary representation. It is a compression method that does not reduce the speed of graph traversal, compared to the version without common subtrees merging implementation.



a) A grid of voxels b) An SVDAG as a graph c) The final binary representation of SVDAG

Figure 5. An example of the binary representation of three-dimensional space using a Sparse Voxel Directed Acyclic Graph (SVDAG) and an z-order curve linearization

The disadvantage of the SVDAG data structure, in comparison to SVO, is that it consumes large amounts of secondary storage or memory. Due to the unused 24 bits used for aligning the child node mask (CHNM) to 32 bits, compared to the 8 bits of SVO. Another disadvantage may be the introduction of 32 b child node pointers whose binary representation may be unnecessarily long – these are not present in pointerless SVOs. However, this disadvantage compensates multiple referencing to common subtrees. It can offset these disadvantages to the extent that for a particular three-dimensional grid, the binary representation of an SVDAG may be more compact than that of an SVO.

Figure 5 a) shows an example of a three-dimensional voxel grid, while Figure 5 b) shows a Sparse Voxel Directed Acyclic Graph. The binary representation of SVDAG shown in Figure 5 c), where the first node is decomposed and mask for its child nodes is "00001001", with two non-empty octants, represented by bits 1. For each non-empty node, there is a 32-bit address (CHNA0 and CHNA3) pointing to the memory location where the child node binary representation is stored. Since those nodes are identical, both pointers are referencing the same address in memory. Another node, referenced two times, has the mask "00001101", i.e., it has three active octants. That is why three addresses (CHNA0, CHNA2, and CHNA3) are present for the child nodes (without actual addresses in this example).

4 POINTERLESS SPARSE VOXEL DIRECTED ACYCLIC GRAPH

The hierarchical data structure, the Pointerless Sparse Voxel Directed Acyclic Graph (PSVDAG), proposed in this paper, is a directed acyclic graph. It represents a three-dimensional grid of voxels R, organized as $N \times N \times N$ voxels. $R = N^3$, where $N = 2^n$,

$n \in \mathbb{N}$. Each voxel is represented by 1 b, when active voxel is encoded by the value '1' and passive voxel by the value '0'. For storing the grid R in uncompressed form, there is a need for the space, which size S is:

$$S = 2^{3n} \text{ [b]}. \tag{2}$$

When represented by PSVDAG hierarchical data structure, the grid R mentioned above is encoded into nodes stored in n levels. Each level has a level mark l with the value from the range of $\langle 0; n-1 \rangle$. The root node is in the level, where $l = 0$. Nodes stored in levels which level mark $l \leq n-2$ have the same structure, described in the Section 4.1 – these are the Inner nodes. Leaf nodes, i.e., nodes stored in the level $l = n-1$, have a different structure, described in Section 4.2.

Formal description of the PSVDAG using Backus-Naur Form (BNF) is as follows:

PSVDAG ::= ⟨NODE⟩
NODE ::= ⟨INODE⟩ | ⟨LNODE⟩
INODE ::= ⟨ACHNC⟩⟨CHNM⟩
ACHNC ::= (3)⟨BIT⟩
CHNM ::= (1) ∗ (8)⟨HT⟩
HT ::= "00" | "01"⟨LAB⟩⟨NODE⟩ | "10"⟨CAL⟩ | "11"⟨NODE⟩
LAB ::= ⟨SIZ⟩⟨VAL⟩
CAL ::= ⟨SIZ⟩⟨VAL⟩
SIZ ::= (5)⟨BIT⟩
VAL ::= (1) ∗ (32)⟨BIT⟩
LNODE ::= (8)⟨BIT⟩
BIT ::= "0" | "1"

where following is applied:

- ⟨SYM⟩ – mandatory symbol SYM,
- "sym" – terminal symbol *sym*,
- $(n)\langle \text{SYM} \rangle$ – symbol SYM concatenated n times,
- $(n) * (m)\langle \text{SYM} \rangle$ – symbol SYM concatenated from n to m times,
- | – alternative,
- *juxtaposition* – the concatenation.

An example in Figure 6 is, for the sake of simplicity, based on the two-dimensional grid of 4×4 pixels. Related quadtree is constructed along with the PSVDAG, using two different space-filling curves (z-order and Hilbert) for linearization.

Formal description of the PSVDAG for two-dimensional grid includes the following changes:

ACHNC ::= (2)⟨BIT⟩
CHNM ::= (1) ∗ (4)⟨HT⟩
LNODE ::= (4)⟨BIT⟩

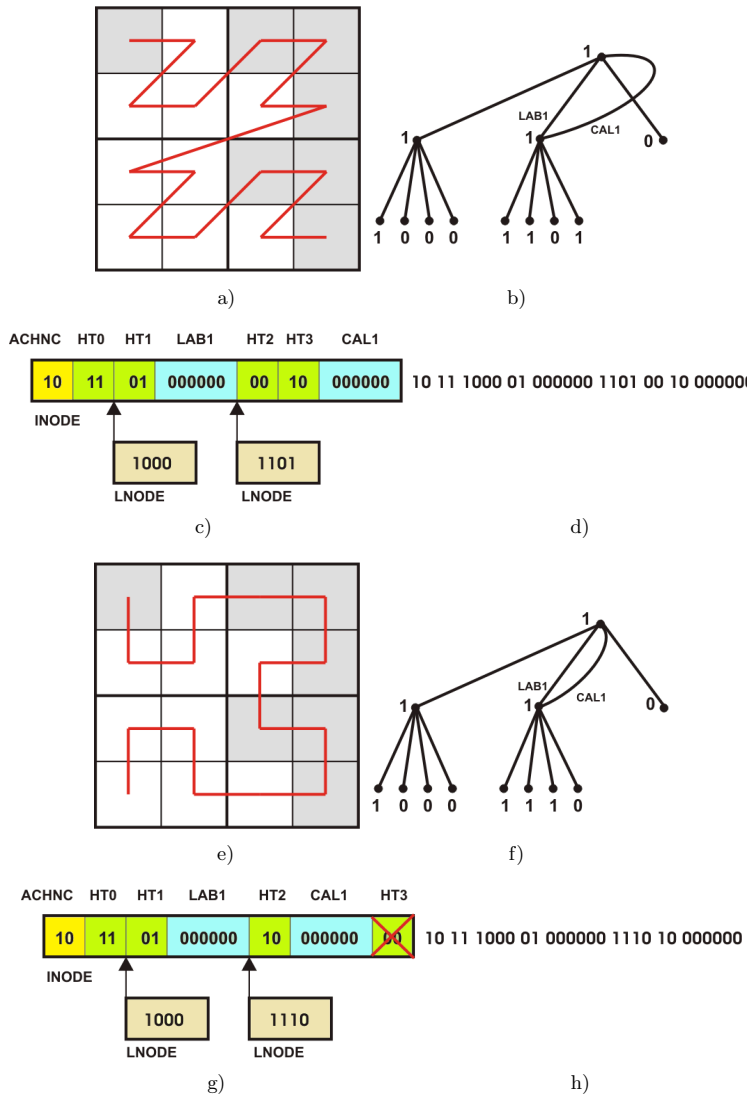


Figure 6. An example of a) 2D grid of pixels and z-order space-filling curve, b) related SVDAG, c) PSVDAG, and, d) final binary representation of PSVDAG. An example of e) 2D grid of pixels and Hilbert space-filling curve, f) related SVDAG, g) PSVDAG, and, h) final binary representation of PSVDAG. ACHNC – Active Child Node Count, HTi – Header Tag, LAB1 – Label, CAL1 – Caller, INODE – Inner Node, LNODE – Leaf Node. SVDAG includes denotations LAB1 and CAL1 to describe the relationship to PSVDAG construction better.

4.1 Nodes

Node of the data structure represents grid $R' : R' \subseteq R$ of voxels, where $R' = N'^3$, $N' = 2^m$, $m \in \mathbb{N} \wedge m \leq n$. There is possibility to find $t = N'/2$ and R' can be divided into 8 octants, forming the set OCT:

$$\text{OCT} = \{oct_0, oct_1, oct_2, \ldots, oct_5, oct_6, oct_7\} \tag{3}$$

where

$$\begin{aligned}
oct_0 &= \langle 0; t-1\rangle \times \langle 0; t-1\rangle \times \langle 0; t-1\rangle,\\
oct_1 &= \langle t; N'-1\rangle \times \langle 0; t-1\rangle \times \langle 0; t-1\rangle,\\
oct_2 &= \langle 0; t-1\rangle \times \langle t; N'-1\rangle \times \langle 0; t-1\rangle,\\
oct_3 &= \langle t; N'-1\rangle \times \langle t; N'-1\rangle \times \langle 0; t-1\rangle,\\
oct_4 &= \langle 0; t-1\rangle \times \langle 0; t-1\rangle \times \langle t; N'-1\rangle,\\
oct_5 &= \langle t; N'-1\rangle \times \langle 0; t-1\rangle \times \langle t; N'-1\rangle,\\
oct_6 &= \langle 0; t-1\rangle \times \langle t; N'-1\rangle \times \langle t; N'-1\rangle,\\
oct_7 &= \langle t; N'-1\rangle \times \langle t; N'-1\rangle \times \langle t; N'-1\rangle.
\end{aligned}$$

Octants $oct_i \in \text{OCT} : i \in \langle 0; 7\rangle$ are ordered in the OCT set according to selected space-filling curve.

When $R' = R$; $m = n$, the root node of the PSVDAG is constructed.

When $m > 1$, the node is constructed as the Inner node INODE.

When $m = 1$, the node is constructed as the Leaf node LNODE.

4.2 Inner Nodes

Inner node consists of Active Child Node Count (ACHNC) followed by the Child Node Mask (CHNM):

INODE ::= ⟨ACHNC⟩⟨CHNM⟩

4.2.1 Active Child Node Count

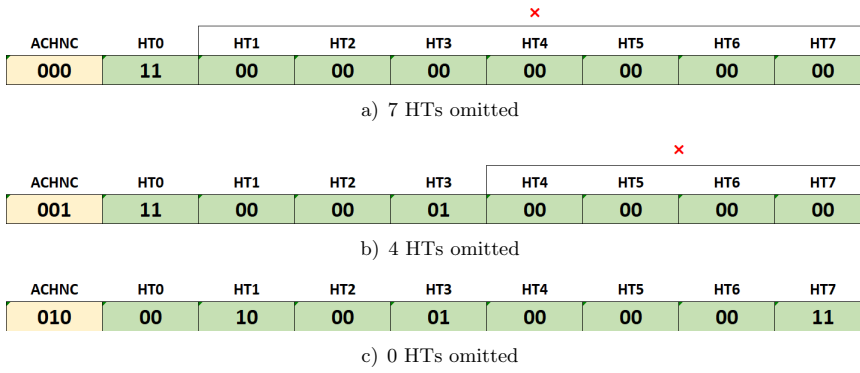
Inner node includes Active Child Node Count (ACHNC) information:

ACHNC ::= (3)⟨*BIT*⟩

It is a 3-bit vector forming unsigned integer number that is referring to the number of active child nodes of a given node. Active child nodes are those having identification '01', '10' or '11' in their HTs, that are part of Inner nodes Child Node Mask (CHNM). An Inner Node cannot have 0 active child nodes, therefore count

range is $\langle 1; 8\rangle$. Decrementing the number of active child nodes by the value 1 allows us to achieve the ACHNC range of $\langle 0; 7\rangle$ to represent it on 3 bits. Thus, if the count of active child nodes is c, the binary representation of ACHNC is set to an unsigned integer of the value $c - 1$.

When sequentially browsing the HTs of the particular nodes CHNM from the HT at position 0 (HT0) to the HT at position 7 (HT7), ACHNC allows us to identify last active child node HTx in their sequence. All subsequent HT identifications, i.e., HT$z : z > x$ must be set to '00'. It is possible to derive this information from the ACHNC and the sequence of previous HTs. That is why there is no need to represent HT$z : z > x$ in the CHNM, and it is possible to omit them. It allows to compress the CHNM (See Figure 7).



a) 7 HTs omitted

b) 4 HTs omitted

c) 0 HTs omitted

Figure 7. Examples of ACHNC and concatenated CHNM containing HTs (for simplicity represented only by their identifications) indicate

There is an unfavorable side effect in the time of reconstructing the pointers or converting PSVDAG to SVDAG. The individual HTs identifications in the CHNM of the PSVDAG node, in general, do not form a continuous vector of bits in the linearized binary representation. Therefore, the ACHNC has been added to the data structure to facilitate and accelerate pointers reconstruction. When the processing of a particular PSVDAG node begins, the ACHNC can be used to determine the number of active child nodes of the SVDAG node. Subsequently, also to determine the space needed to store related pointers. Another advantage is that there is a possibility to determine when the last active child node HT has been processed and immediately finish the node processing.

4.2.2 Child Node Mask

Child node mask consists of Header Tags (HT), HT$_i : i \in \langle 0; 7\rangle$, which are assigned to octants oct$_i \in$ OCT $: i \in \langle 0; 7\rangle$ in ascending order. Encoding of the CHNM and HT is as follows:

CHNM ::= $(1) * (8)\langle \text{HT} \rangle$
HT ::= "00" | "01"$\langle \text{LAB} \rangle\langle \text{NODE} \rangle$ | "10"$\langle \text{CAL} \rangle\rangle$ | "11"$\langle \text{NODE} \rangle$

Passive child node HT. Octant $oct_i \in \text{OCT} : i \in \langle 0; 7 \rangle$ is passive when it does not contain any active voxels. The HT is, in that case, represented only by identification symbol "00". No further information follows (Passive child node header tags can be omitted, as mentioned above in the Section 4.2.1).

Active child node HT. Octant $oct_i \in \text{OCT} : i \in \langle 0; 7 \rangle$ is active when its grid contains at least one active voxel.

If octant $oct_i \in \text{OCT} : i \in \langle 0; 7 \rangle$ is not unique (in related SVDAG its node is referenced by more than one pointer), and it will serve as the template, the HT is represented by symbol "01". Label LAB concatenates along with the NODE, constructed for the grid of this octant.

If octant $oct_i \in \text{OCT} : i \in \langle 0; 7 \rangle$ is not unique (in related SVDAG, there is more than one pointer to this node), and it will not serve as the template, the HT is represented by symbol "10". Caller CAL concatenates.

If octant $oct_i \in \text{OCT} : i \in \langle 0; 7 \rangle$ is unique (in related SVDAG, there is only one pointer to this node), the HT is represented by symbol "11". Binary representation of the NODE constructed for the grid of this octant concatenates.

When octant $oct_i \in \text{OCT} : i \in \langle 0; 7 \rangle$ is not unique (it is referenced in related SVDAG x times), its binary representation in PSVDAG is fully developed only once, when it is referenced by the assigned label LAB_d to which the octants NODE is concatenated. The octant is further referenced $x - 1$ times using caller CAL_d. Binary representation of $\text{LAB}_d = \text{CAL}_d$.

That is how the set P of x pointers to the particular common subDAG of SVDAG is replaced by one label LAB_d followed by the binary representation of this subDAG and $x - 1$ times by the caller CAL_d. The first used pointer from the set P, when traversing SVDAG in Depth-First order, is replaced by a label followed by the binary representation of subDAG. Caller will replace the other pointers from the set P. It ensures that when traversing PSVDAG in the Depth-First order, the label LAB_d will be used before any of occurrences of the caller CAL_d – see Figure 8.

4.2.3 Labels and Callers

Labels in the PSVDAG data structure identify the corresponding octants. Binary encoding of the label is respecting the rules:

LAB ::= $\langle \text{SIZ} \rangle\langle \text{VAL} \rangle$
SIZ ::= $(5)\langle \text{BIT} \rangle$
VAL ::= $(1) * (32)\langle \text{BIT} \rangle$
BIT ::= "0" | "1"

SIZ is a 5-bit vector representing an unsigned integer from the range of $\langle 0 : 31 \rangle$. Incremented by value 1, it is representing the number of bits of the concatenated

VAL. Provided the SIZ is 0, the length of the binary representation of the VAL is 1 b.

VAL is the (SIZ + 1) – bit vector, that is representing unsigned integer value from the range of $\langle 0; 2^{SIZ+1} - 1\rangle$.

When set of labels is generated for the PSVDAG data structure, (for each level l separately), the SIZ = 0 and VAL = 0, for the label $\text{LAB}_i : i = 0$. For each next label the VAL is incremented by 1, until the value of the $\text{VAL} = 2^{\text{SIZ}+1} - 1$. For next label VAL = 0 and SIZ is incremented by 1.

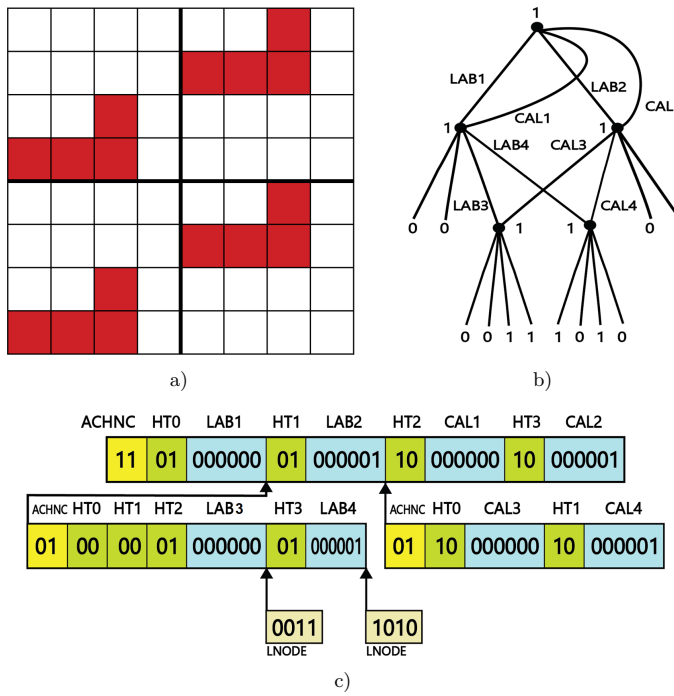


Figure 8. Example of a) two-dimensional grid of 8×8 pixels, b) SVDAG with denotations of LABs and CALs for better description of c) PSVDAG

It is possible to have two different labels with the same SIZ and VAL values, i.e., LAB_a and LAB_b, where $\text{SIZ}_a = \text{SIZ}_b \wedge \text{VAL}_a = \text{VAL}_b$, if those labels are located in different levels l of the data structure. Provided there is l, SIZ, and VAL concatenated, each label is unique in the data structure. There is no need to represent level l value in the PSVDAG because it is possible to derive it from the data structure in the traversing time.

Callers are constructed in the same way as the labels, using the rules:

CAL ::= ⟨SIZ⟩⟨VAL⟩
SIZ ::= (5)⟨BIT⟩

VAL ::= (1) ∗ (32)⟨BIT⟩
BIT ::= "0" | "1"

Caller CAL_C with the level value l_C, SIZ = siz_C, VAL = val_C is referencing the same node as the LAB_L with the level value l_L, SIZ = siz_L, VAL = val_L when $l_C = l_L \wedge siz_C = siz_L \wedge val_C = val_L$.

4.2.4 Frequency-Based Compaction

Particular nodes in PSVDAG data structure at level l are referenced from the level $l-1$ different times, using labels and callers (references). Each label/caller can have variable length of the binary representation. Different permutations of labels/callers assignment to particular nodes generate different overall sum of labels/callers lengths. Therefore frequency-based compaction is applied to find out permutation that has the smallest overall sum of reference lengths.

It is possible to form the NOD set consisting of p nodes from level l of PSVDAG that are each referenced more than once. Members of the set are in descending order according to the frequency of their references.

$$\mathrm{NOD} = \{nod_0, nod_1, nod_2, \ldots, nod_{p-3}, nod_{p-2}, nod_{p-1}\}. \tag{4}$$

Generated labels – unique one for each $nod_i \in \mathrm{NOD} : i \in \langle 0; p-1\rangle$ – form the set LAB that consists of p labels, that have different lengths of their encoding. The LAB set ordering is ascending order according to the length of the binary representation of particular labels.

$$\mathrm{LAB} = \{lab_0, lab_1, lab_2, \ldots, lab_{p-3}, lab_{p-2}, lab_{p-1}\}. \tag{5}$$

Using frequency-based compaction, the node $nod_i \in \mathrm{NOD} : i \in \langle 0; p-1\rangle$ is assigned by the label $lab_j \in \mathrm{LAB} : j \in \langle 0; p-1\rangle$, $i = j$.

Each node assigned by the label $lab_i \in \mathrm{LAB} : i \in \langle 0; p-1\rangle$ with the length of binary representation len is referenced by the caller (used one or more times) that has the same length of the binary representation len.

An example of different permutations of label assignments, including one based on frequency-based compaction, can be seen in Figure 9.

4.3 Leaf Nodes

By the Leaf node, the octant, which grid $R' = 2^3$ (containing 8 voxels), is encoded. Leaf node consists of 8 bit vector where only information about voxels is stored. Passive voxels are represented using bit 0, active voxels are represented using bit 1. Linearization of the grid R' is performed according to chosed space-filling curve, as it is depicted on Figure 10.

Node id	1	2	3	4	5	
Referencing frequency	2	1	3	8	1	
Reference [b]	6	6	7	7	7	
Sum [b]	12	6	21	56	7	∑ = 102

Node id	4	3	1	2	5	
Referencing frequency	8	3	2	1	1	
Reference [b]	6	6	7	7	7	
Sum [b]	48	18	14	7	7	∑ = 94

Figure 9. An example of two different permutations of label assignments to the respective nodes of a particular PSVDAG level (for clarity, Node id is added in this example). The top part of the figure represents arbitrary node order, and the overall sum of labels/callers length is 102 b, the bottom part represents node order achieved using frequency-based compaction, and in this case, the overall sum of labels/callers length is 94 b.

5 RESULTS AND DISCUSSION

Various 3D scenes, originally stored in Wavefront OBJ geometry definition file format (examples of those scenes with their visualizations are in Figure 11), were used for test purposes. Voxelization was performed to the different resolutions ranging from 128^3 to the $4\,096^3$ ($4K^3$). Voxelized scenes were encoded to SVO, SVDAG, and PSVDAG hierarchical data structures. Evaluation of results obtained by tests

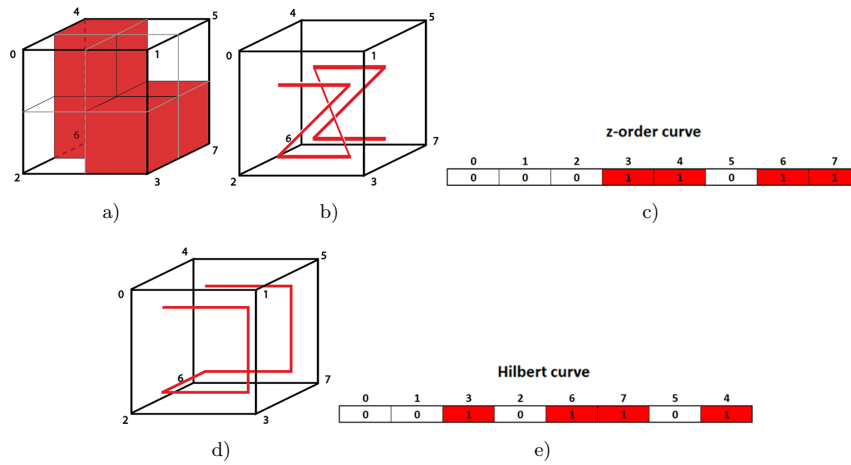
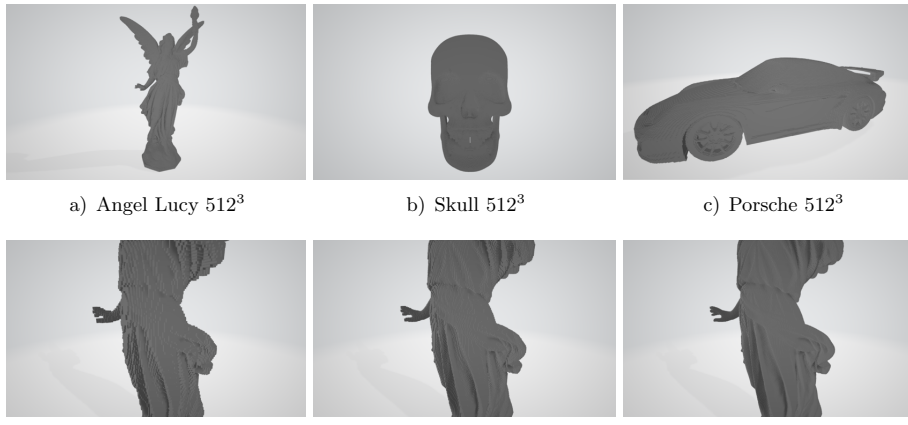


Figure 10. a) Leaf node consisting of 4 active voxels (red), b) z-order curve, c) voxels in binary representation of the node arranged according to z-order curve, d) Hilbert curve, e) voxels in binary representation of the node arranged according to Hilbert curve

followed. Tests were performed on the computer using four core CPU Intel® Core i5-8265U at 1.6 GHz, 8 GB operating memory, NVIDIA GeForce GTX 1050 3 GB graphics card, and 256 GB SSD as the secondary storage.



a) Angel Lucy 512^3 b) Skull 512^3 c) Porsche 512^3

d) Detail of the Angel Lucy 256^3 e) Detail of the Angel Lucy 512^3 f) Detail of the Angel Lucy $1K^3$

Figure 11. Visualization of examples of voxelized scenes used for testing purposes

The test results show that the representation of the scene geometry using the PSVDAG data structure is significantly smaller than in SVDAG. It applies to all tested three-dimensional voxelized data sets and the z-order and Hilbert curve linearizations. In the most optimistic case, at a 128^3 resolution, a compression ratio of 3.77 was reached. The binary representation of this scene in PSVDAG thus represents only 26.5 % of the size of its binary encoding in SVDAG. Generally speaking, an increase in the resolution leads to a gradual decrease in the compression ratio. It reached the value 2.80 at a resolution of $4\,096^3$, when the binary representation in PSVDAG occupies only 35.7 % of the corresponding encoding size in SVDAG.

The size of the binary representation in PSVDAG is, with increasing resolution, relatively decreasing, compared to SVO. In most cases, PSVDAG outperformed SVO for both, z-order curve as well as Hilbert curve linearizations. In these tests, PSVDAG to the corresponding SVO size ratio, is ranging from 131.1 % to 54.5 %. The compression ratio is also shown separately in Figures 12 and 13 for z-order curve linearization.

A comparison in terms of absolute sizes of binary representations is available in Table 1 for z-order curve linearization and in Table 2 for the Hilbert curve. The PSVDAG-SVDAG and PSVDAG-SVO compression ratios are available in Tables 3 and 5. This also shows the relative space requirements of the PSVDAG-encoded scene, compared to the SVDAG and SVO versions. The PSVDAG-SVDAG compression ratio, when the z-order curve is applied, is comparable to the version using the Hilbert curve, for all models and resolutions. There are only small fluctuations in the range of $\langle -0.687\,\%; 1.657\,\% \rangle$. Size comparison showed a slightly higher success

Size [KB]	Resolution					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
SVO	6.67	28.62	117.99	475.98	1 895.02	7 447.62
SVDAG	31.88	127.01	462.85	1 523.74	4 682.54	14 928.34
PSVDAG	8.62	35.10	132.37	462.04	1 530.86	5 208.41
Skull						
SVO	23.24	95.61	387.45	1 551.55	6 129.76	23 667.56
SVDAG	100.39	356.75	1 182.45	3 728.80	12 275.23	43 086.88
PSVDAG	28.11	104.31	366.48	1 239.06	4 320.48	15 279.20
Porsche						
SVO	14.59	67.52	295.10	1 241.50	5 087.54	20 262.88
SVDAG	56.78	222.41	788.76	2 629.21	8 918.93	32 127.96
PSVDAG	15.08	61.28	227.13	800.89	2 894.32	11 048.85

Table 1. Comparison of SVO, SVDAG and PSVDAG hierarchical data structures using different scenes and resolutions in terms of data structure size (their binary representation in KB) – linearized using z-order curve

in common subtree merges when z-order linearization is applied. Subsequently, the memory consumption and the time for the compression of SVDAG data structure were also smaller, as shown in Tables 7 and 8.

Adding Active Child Node Count information allows potential compression of the Child Node Mask. Hence, the authors monitored whether the 16 b of identifications of Child Node Mask was compressed or inflated in the tested scenes. The

Size [KB]	Resolution					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
SVO	6.67	28.62	117.99	475.98	1 895.02	7 447.62
SVDAG	32.14	128.25	477.07	1 596.11	4 922.56	15 519.44
PSVDAG	8.74	36.00	137.74	486.17	1 613.09	5 434.41
Skull						
SVO	23.24	95.61	387.45	1 551.55	6 129.76	23 667.56
SVDAG	101.55	362.38	1 197.48	3 780.68	12 447.23	43 745.21
PSVDAG	28.44	105.87	371.02	1 254.54	4 370.65	15 621.87
Porsche						
SVO	14.59	67.52	295.10	1 241.50	5 087.54	20 262.88
SVDAG	59.99	234.71	836.33	2 802.36	9 480.74	34 082.36
PSVDAG	15.92	64.75	240.71	852.27	3 064.59	11 637.51

Table 2. Comparison of SVO, SVDAG and PSVDAG hierarchical data structures using different scenes and resolutions in terms of data structure size (their binary representation in KB) – linearized using Hilbert curve

Parameter	**Resolution**					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
PSVDAG in comparison to SVO						
Compression	0.77	0.82	0.89	1.03	1.24	1.43
Percents	129.3 %	122.6 %	112.2 %	97.1 %	80.8 %	69.9 %
PSVDAG in comparison to SVDAG						
Compression	3.70	3.62	3.50	3.30	3.06	2.87
Percents	27.0 %	27.6 %	28.6 %	30.3 %	32.7 %	34.9 %
Skull						
PSVDAG in comparison to SVO						
Compression	0.83	0.92	1.06	1.25	1.42	1.55
Percents	120.9 %	109.1 %	94.6 %	79.9 %	70.5 %	64.6 %
PSVDAG in comparison to SVDAG						
Compression	3.57	3.42	3.23	3.01	2.84	2.82
Percents	28.0 %	29.2 %	31.0 %	33.2 %	35.2 %	35.5 %
Porsche						
PSVDAG in comparison to SVO						
Compression	0.97	1.10	1.30	1.55	1.76	1.83
Percents	103.4 %	90.7 %	77.0 %	64.5 %	56.9 %	54.5 %
PSVDAG in comparison to SVDAG						
Compression	3.76	3.63	3.47	3.28	3.08	2.91
Percents	26.6 %	27.6 %	28.8 %	30.5 %	32.5 %	34.4 %

Table 3. Comparison of compression ratios and size percentages of binary representations of PSVDAG to SVO and PSVDAG to SVDAG hierarchical data structures, using different scenes and resolutions – linearized using z-order curve

tests showed that the size of the child node masks binary representation increased in most cases – its average length is from the range of $\langle 15.82; 17.34\rangle$ b. Thus, in some models, the average size of the child node mask was reduced up to 98.88 % in comparison to 16 b size. In some models, inflation was evident (in this case, the most pessimistic case showed a 108.38 % compared to the 16 b size.) However, even in the most pessimistic case, the combined size of ACHNC and CHNM was smaller than the CHNM of the SVDAG data structure, accompanied by the reserved 24 b. In this case, the worst compression ratio reached for this part of the node was 1.85 (54.19 % of the original size). The relevant data is available in Tables 4, 6, and Figure 14.

The average length of the binary representation of labels/callers gradually increases with an increased resolution of the voxelized 3D scene because of the rising number of labels. For example, at the lowest resolution of 128^3, the Angel Lucy model for z-order averaged 8.75 b per label/caller, and 8.90 b in case of Hilbert curve. It is a 3.66 (27.34 %), resp. 3.60 (27.81 %) compression ratio, compared to the 32 b SVDAG pointers. The maximum length of the label/caller binary repre-

Size [b]	Resolution					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
Child Node Mask	15.89	16.73	17.11	17.28	17.33	17.30
Labels and Callers	8.75	8.90	9.21	9.86	10.87	11.93
Skull						
Child Node Mask	16.82	17.12	17.29	17.35	17.33	17.34
Labels and Callers	9.01	9.43	10.11	11.06	12.03	12.49
Porsche						
Child Node Mask	16.89	17.23	17.43	17.44	17.39	17.32
Labels and Callers	8.24	8.66	9.17	9.91	10.83	11.79

Table 4. Selected parameters of PSVDAG hierarchical data structure according to the different scenes and their resolutions – linearized using z-order curve

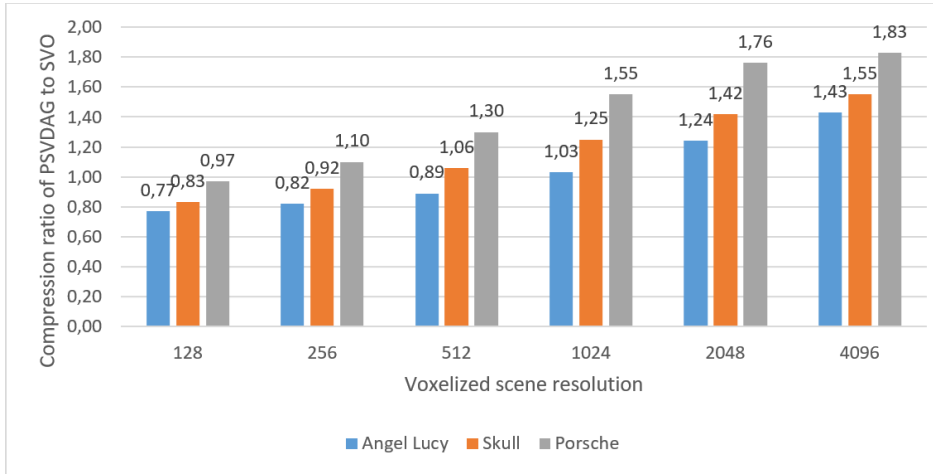


Figure 12. The PSVDAG-SVO compression ratio for each scene and the resolutions used to voxelize the respective scenes, when z-order applied

sentation for this model reached a maximum at the resolution of $4\,096^3$ when the average label/caller size was 11.93 b resp. 11.96 b. It represents a 2.68-fold compression compared to the 32 b SVDAG pointers and reduction to 37.28 % resp. 37.38 % in terms of size. The numerical values of this parameter for the respective models and resolutions are in Tables 4, 6, and Figure 15.

5.1 Sources of Compression Gains in PSVDAG

The PSVDAG data structure provides several data compression sources, when compared to the SVDAG:

Parameter	**Resolution**					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
PSVDAG in comparison to SVO						
Compression	0.76	0.80	0.86	0.98	1.17	1.37
Percents	131.1 %	125.8 %	116.7 %	102.1 %	85.1 %	73.0 %
PSVDAG in comparison to SVDAG						
Compression	3.68	3.56	3.46	3.28	3.05	2.86
Percents	27.2 %	28.1 %	28.9 %	30.5 %	32.8 %	35.0 %
Skull						
PSVDAG in comparison to SVO						
Compression	0.82	0.90	1.04	1.24	1.40	1.52
Percents	122.4 %	110.7 %	95.8 %	80.9 %	71.3 %	66.0 %
PSVDAG in comparison to SVDAG						
Compression	3.57	3.42	3.23	3.01	2.85	2.80
Percents	28.0 %	29.2 %	31.0 %	33.2 %	35.1 %	35.7 %
Porsche						
PSVDAG in comparison to SVO						
Compression	0.92	1.04	1.23	1.46	1.66	1.74
Percents	109.1 %	95.9 %	81.6 %	68.6 %	60.2 %	57.4 %
PSVDAG in comparison to SVDAG						
Compression	3.77	3.62	3.47	3.29	3.09	2.93
Percents	26.5 %	27.6 %	28.8 %	30.4 %	32.3 %	34.1 %

Table 5. Comparison of compression ratios and size percentages of binary representations of PSVDAG to SVO and PSVDAG to SVO hierarchical data structures, using different scenes and resolutions – linearized using Hilbert curve

- The child nodes mask of the SVDAG data structure has a constant size of 8 b (i.e., 1 b per octant), with another 24 b unused for the sake of aligning to 32 b. That results in a total of 32 b, i.e., 4 b per octant. PSVDAG uses a 3-bit Active Child Node Count. The Child Node Mask identifications itself have a binary representation with a total length ranging from 2 b to 16 b, with 2 b per potential child node. The sum of ACHNC and CHNM ranges from 5 b to 19 b, which is significantly less than that of SVDAG. 8 octants use space ranging from 0.625 b per child node to 2.375 b per child node. The compression ratio (in comparison to SVDAG) is from the range of 1.68 to 6.4. The reduction of this part of the data structure is to a level of 15.63 % to 59.38 %.
- The absence of child node pointers in PSVDAG is a significant source of its reduction, considering the binary representation (compared to the SVDAG data structure). If the subgraph is unique, there is no pointer at its root node in the data structure. Thus the 32 b pointer of SVDAG becomes 0 b in PSVDAG.
- If the particular subgraph is not unique in the data structure, the 32 b pointer of the SVDAG, pointing at the root node of the subgraph, is replaced by a la-

Size [b]	Resolution					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
Child Node Mask	15.82	16.69	17.08	17.28	17.34	17.32
Labels and Callers	8.90	9.12	9.37	9.95	10.90	11.96
Skull						
Child Node Mask	16.75	17.07	17.26	17.30	17.27	17.27
Labels and Callers	9.04	9.44	10.11	11.05	12.00	12.41
Porsche						
Child Node Mask	16.80	17.14	17.31	17.35	17.30	17.25
Labels and Callers	8.30	8.71	9.21	9.92	10.80	11.70

Table 6. Selected parameters of PSVDAG hierarchical data structure according to the different scenes and their resolutions – linearized using Hilbert curve

Size [KB]	Resolution					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
z-order curve	39.48	156.13	544.49	1 635.08	4 225.91	10 801.93
Hilbert curve	39.74	157.82	566.83	1 752.33	4 617.10	11 745.66
Skull						
z-order curve	120.63	406.15	1 224.49	3 244.12	8 718.88	33 376.02
Hilbert curve	122.30	414.52	1 245.42	3 320.14	8 967.54	34 912.34
Porsche						
z-order curve	72.73	263.09	857.53	2 492.10	7 142.92	22 457.98
Hilbert curve	77.91	281.10	919.52	2 714.14	7 892.19	25 043.80

Table 7. Memory consumption for PSVDAG building and encoding

bel/caller with at least 6 b binary representation. As the number of labels/callers in the data structure increases, this binary representation length gradually increases. Thus, the shortest ones represent a 5.33-fold compression rate, or, in other words: occupy only 18.75 % of the space.

- Minimization of the label/caller binary representation length is supported by separating labels assignments at every level of the graph, instead of assigning labels globally within the entire data structure. Thus, the label assignment starts at every level of the data structure with a length of the 6 b.
- Frequency-based compaction sorts the subgraph root nodes separately at each level l of the graph, according to their frequency of referencing from the level $l-1$. The process sorts labels/callers from the most frequently referenced nodes to the least frequently referenced ones. Individual root nodes of the template subgraphs are then assigned by labels, in the order from those having the shortest binary representation (6 b – attached to the most frequently referenced nodes) to those having the most extended binary representation (assigned to the least often referenced subgraph root nodes). It ensures the lowest possible number of

Time [s]	**Resolution**					
	128^3	256^3	512^3	$1K^3$	$2K^3$	$4K^3$
Angel Lucy						
z-order curve	0.021	0.032	0.079	0.196	0.402	1.279
Hilbert curve	0.021	0.033	0.081	0.205	0.423	1.331
Skull						
z-order curve	0.068	0.091	0.202	0.480	1.051	4.065
Hilbert curve	0.068	0.092	0.205	0.486	1.065	4.154
Porsche						
z-order curve	0.039	0.057	0.136	0.339	0.765	2.744
Hilbert curve	0.041	0.061	0.144	0.362	0.813	2.912

Table 8. Time consumption for PSVDAG building and encoding

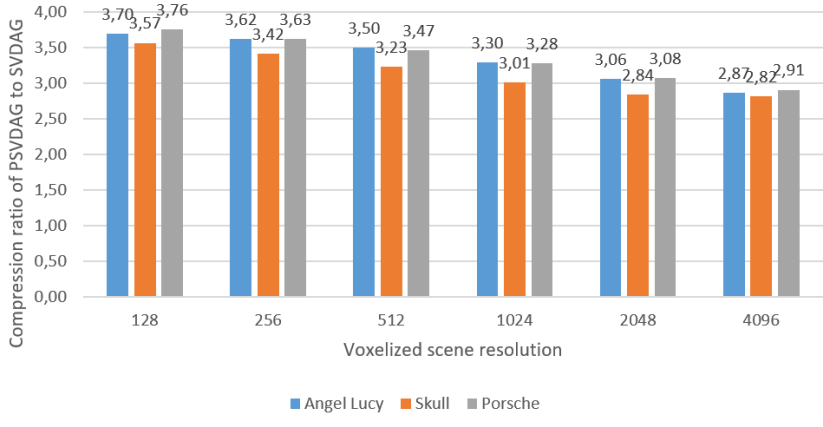


Figure 13. The PSVDAG-SVDAG compression ratio for each scene and the resolutions used to voxelize the respective scenes, when z-order applied

bits for the encoding of labels/callers per level of the graph.

6 CONCLUSIONS

This paper deals with hierarchical data structures designed to represent the geometry of voxelized three-dimensional scenes. It includes an overview of the related works published in the field of linearization of multi-dimensional data and the representation of two and three-dimensional data using dedicated data structures. The brief presentation of pointerless Sparse Voxel Octrees (SVO) and Sparse Voxel Directed Acyclic Graphs (SVDAG) summarized their advantages, disadvantages, and binary-level encoding. While pointerless SVO is a compact, pointerless data structure that is suitable for archival or streaming purposes, it is necessary to create

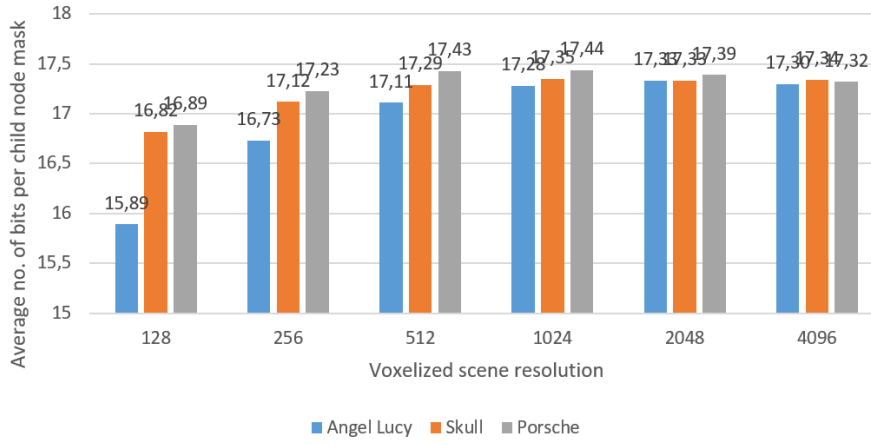


Figure 14. Average Child Node Mask size in bits for the scenes and the resolutions used to voxelize the respective scenes, when z-order applied

child node pointers for traversal. SVDAG is a more advanced data structure that incorporates 32 b child node pointers to allow fast traversal while reducing common subtrees.

In the subsequent section, the authors presented the proposed Pointerless Sparse Voxel Directed Acyclic Graph (PSVDAG) data structure, which combines the advantages of both data structures mentioned above. It allows a compact encoding

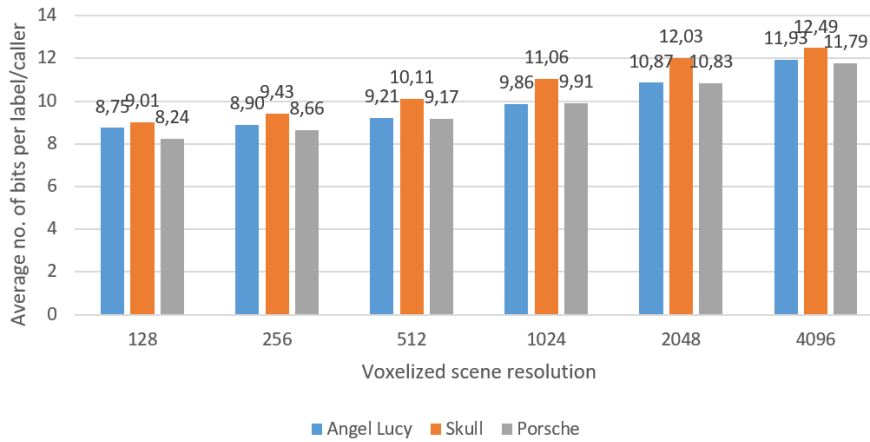


Figure 15. Average label/caller size in bits for the scenes and the resolutions used to voxelize the respective scenes, when z-order applied

of the data, and merging of common subtrees using the proposed concept of labels and callers having a variable-length binary representation. Compared to SVDAG, the disadvantage is that for fast traversal, this data structure requires the reconstruction of pointers to child nodes. The Active Child Node Count proposed in this paper facilitates and speeds up this process. In a favorable but less frequent case, it enables child node mask compression.

Performed tests were using scenes initially represented in the OBJ file format, and subsequently voxelized at various resolutions. The obtained results showed that – compared to the SVDAG data structure – PSVDAG achieved a compression ratio of 3.77 to 2.80 times higher. In most cases, PSVDAG outperformed the SVO, when the size of PSVDAG ranged from 131.1 % of the size of SVO (in the most unfavorable case) to 54.5 % of the size of SVO (in the most favorable case). Thus, in favorable circumstances, the binary representation of PSVDAG was more compact than that of SVO.

Acknowledgements

This research was supported by the Slovak Research and Development Agency, project number APVV-18-0214 and the KEGA Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under Grant No. 003TUKE-4/2017 Implementation of Modern Methods and Education Forms in the Area of Security of Information and Communication Technologies towards Requirements of Labor Market.

REFERENCES

[1] Crassin, C.—Neyret, F.—Lefebvre, S.—Eismann, E.: GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09), ACM, 2009, pp. 15–22, doi: 10.1145/1507149.1507152.

[2] Balsa Rodríguez, M.—Gobetti, E.—Iglesias Guitián, J. A.—Makhinya, M.—Marton, F.—Pajarola, R.—Suter, S. K.: State-of-the-Art in Compressed GPU-Based Direct Volume Rendering. Computer Graphics Forum, Vol. 33, 2014, No. 6, pp. 77–100, doi: 10.1111/cgf.12280.

[3] Katajainen, J.—Mäkinen, E.: Tree Compression and Optimization with Applications. International Journal of Foundations of Computer Science, Vol. 1, 1990, No. 4, pp. 425–447, doi: 10.1142/S0129054190000291.

[4] Laszloffy, A.—Long, J.—Patra, A. K.: Simple Data Management, Scheduling and Solution Strategies for Managing the Irregularities in Parallel Adaptive *hp* Finite Element Simulations. Parallel Computing, Vol. 26, 2000, pp. 1765–1788, doi: 10.1016/S0167-8191(00)00054-5.

[5] Sagan, H.: Space-Filling Curves. Springer-Verlag, New York, 1994, doi: 10.1007/978-1-4612-0871-6.

[6] Hilbert, D.: Über die Stetige Abbildung Einer Linie auf ein Flächenstück. Mathematische Annalen, Vol. 38, 1891, pp. 459–460, doi: 10.1007/BF01199431.

[7] 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, 1966.

[8] 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.

[9] 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.

[10] 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.

[11] 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, Vol. PAMI-2, 1980, No. 1, pp. 27–35, doi: 10.1109/TPAMI.1980.4766967.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Jackins, C. L.—Tanimoto, S. L.: Oct-Trees 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.

[18] 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.

[19] Meagher, D.: 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.

[20] 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.

[21] Samet, H.: Implementing Raytracing with Octrees and Neighbor Finding. Computers and Graphics, Vol. 13, 1989, No. 4, pp. 445–460, doi: 10.1016/0097-8493(89)90006-X.

[22] Knoll, A.—Wald, I.—Parker, S. G.—Hansen, C. D.: Interactive Isosurface Ray Tracing of Large Octree Volumes. 2006 IEEE Symposium on Interactive Ray Tracing, Salt Lake City, UT, USA, 2006, pp. 115–124, doi: 10.1109/RT.2006.280222.

[23] Knoll, A. M.—Wald, I.—Hansen, C. D.: Coherent Multiresolution Isosurface Ray Tracing. The Visual Computer, Vol. 25, 2009, No. 3, pp. 209–225, doi: 10.1007/s00371-008-0215-2.

[24] Schnabel, R.—Klein R.: Octree-Based Point Cloud Compression. Proceedings of the 3rd Eurographics Symposium on Point-Based Graphics/IEEE VGTC Conference on Point-Based Graphics (SPBG '06), 2006, pp. 111–121, doi: 10.2312/SPBG/SPBG06/111-120.

[25] Laine, S.—Karras, T.: Efficient Sparse Voxel Octrees. Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '10), ACM, 2010, pp. 55–63, doi: 10.1145/1730804.1730814.

[26] Kämpe, V.—Sintorn, E.—Assarsson, U.: High Resolution Sparse Voxel DAGs. ACM Transactions on Graphics, Vol. 32, 2013, No. 4, Art. No. 101, 8 pp., doi: 10.1145/2461912.2462024.

[27] Villanueva, A. J.—Marton, F.—Gobbetti, E.: SSVDAGs: Symmetry-Aware Sparse Voxel DAGs. Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '16), ACM, 2016, pp. 7–14, doi: 10.1145/2856400.2856420.

[28] Čerešník, P.—Madoš, B.—Baláž, A.—Bilanová, Z.: SSVDAG*: Efficient Volume Data Representation Using Enhanced Symmetry-Aware Sparse Voxel Directed Acyclic Graph. Proceedings of the 2019 IEEE 15th International Scientific Conference on Informatics, IEEE, 2019, pp. 70–75, doi: 10.1109/Informatics47936.2019.9119298.

[29] Baert, J.—Lagae, A.—Dutré, P.: Out-of-Core Construction of Sparse Voxel Octrees. Proceedings of the 5th High-Performance Graphics Conference (HPG '13), ACM, 2013, pp. 27–32, doi: 10.1145/2492045.2492048.

Liberios Vokorokos graduated (M.Sc.) with honors at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice in 1991. He defended his Ph.D. in the field of programming devices and systems in 2000 with the thesis title "Diagnosis of Compound Systems Using the Data Flow Applications". He serves as Professor for computer science and informatics since 2005. Since 1995 he is working as Educationist at the Department of Computers and Informatics. His scientific research focuses on parallel computers of the data flow type. He also investigates the questions related to the diagnostics of complex systems. Currently, he is the Dean of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. He is a member of the Advisory Committee for Informatization at the Faculty and Advisory Board for the Development and Informatization at the Technical University of Košice.

Branislav Madoš graduated (Ing.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2006. He defended his Ph.D. in the field of computers and computer systems in 2009. His thesis title was "Specialized Architecture of Data Flow Computer". Since 2010 he is Assistant Professor at the Department of Computers and Informatics. His scientific research focuses on parallel computer architectures and architectures of computers with a data-driven computational model, and on computer security using cryptographic and steganographic methods.

Zuzana Bilanová graduated (Ing.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2015. Since 2015 she is Ph.D. student at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. Her main scientific focus is creating new approaches in logical analysis of natural language, non-classical logical systems in computer science, and resource-oriented logic programming. Her secondary research areas are educational technologies for the effective implementation of engineering education, focusing on project and team-based teaching.