IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 4, 2020, accepted July 26, 2020, date of publication July 31, 2020, date of current version August 13, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3013300*

# Visual Hull Tree: A New Progressive Method to Represent Voxel Data

**TAE YOUNG JANG[1], (Member, IEEE), SEONG DAE KIM[1], (Life Senior Member, IEEE), AND SUNG SOO HWANG[2], (Member, IEEE)**

[1]Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea
[2]School of Computer Science and Electrical Engineering, Handong Global University, Pohang 37554, South Korea

Corresponding author: Seong Dae Kim (sdkim@kaist.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2016R1D1A3B03934808.

**ABSTRACT** A visual hull is an approximation of a three-dimensional (3-D) object generated by the shape-from-silhouettes (SfS) technique. Because a visual hull is calculated from silhouettes, a visual hull can be represented by silhouette images, encoded by a small number of bits. However, it is challenging to represent the concave regions of a visual hull. In this paper, we model voxel data with a set of visual hulls, thereby handling the concave regions of voxel data. To accomplish this, silhouettes are generated from input voxel data using virtual cameras, and a visual hull is computed by SfS using the silhouettes. To handle the concave regions, we calculate the residuals of visual hulls, and the residuals are represented by visual hulls again. This process is repeated until all concave regions are processed, and a hierarchical data structure, i.e., a visual hull tree, is generated. Because the visual hull tree is constituted from a set of visual hulls, it can represent the details of the voxel data even in the root node. Also, because a set of visual hulls can be represented by silhouettes, a visual hull tree has a small number of bits. From the experiments, we compare our method to the octree-based representation, and our method demonstrates good encoding performance.

**INDEX TERMS** Three-dimensional data, voxel representation, visual hull, hierarchical representation.

## I. INTRODUCTION

To represent three-dimensional (3-D) geometry data efficiently, a variety of methods have been proposed. Among them, volumetric approaches use voxels as primitives and represent a 3-D geometry using transparent voxels and opaque voxels. Compared to other approaches such as polygonal meshes, volumetric approaches are intuitive and straightforward. Moreover, many signal processing algorithms can be applied to voxel data by considering a voxel as a 3-D extension of a pixel. For these reasons, recent research in computer graphics and artificial intelligence has widely utilized volumetric approaches.

The disadvantage of volumetric approaches is memory requirements, i.e., they generally require a large amount of memory to represent a 3-D geometry. Consequently, several encoding methods for voxel data have been studied, and an octree has become a popular means to represent voxel data [1]–[3]. An octree is the 3-D extension of a quad-tree, and it recursively subdivides a 3-D space into eight octants. When the geometry of a 3-D object is represented by an octree, the shape becomes more apparent as we traverse down the tree.

In this paper, we propose a visual hull tree to represent voxel data. A visual hull is a geometric entity created by the shape-from-silhouettes technique (SfS) [4]–[6]. SfS is a 3-D geometry reconstruction method using multiple-view silhouettes of an object, and computes a visual hull using a small number of computations compared with other 3-D reconstruction methods. From the above observation, we represent voxels as a set of silhouettes used for computing a visual hull using SfS. This implies that while the majority of previous research uses SfS as a tool for generating 3-D contents, we instead use it as a 3-D geometry decoder. However, a visual hull cannot present the concave regions of an object well. To address this issue, we find the concave regions based on the differences between the 3-D geometry and its visual hull. Then, the concave regions are again represented by visual hulls. This process is performed repeatedly, thereby representing voxel data as a set of visual hulls. From the above process, visual hulls are generated hierarchically, which we refer to as a visual hull tree, which is similar to a concavity

The associate editor coordinating the review of this manuscript and approving it for publication was Sudipta Roy.

tree [7]–[9]. Using this method, voxel data is represented by a set of silhouettes that are used for generating visual hulls, and the silhouettes can be encoded by various methods, i.e., 2-D run lengths, a chain code, or quad-tree. However, traditional methods only encode a single silhouette; thus, a new method for representing multiple silhouettes is necessary to reduce data. To address this, we represent a set of silhouettes using the bit plane approach. The bit plane method converts silhouettes to gray-scale images and it can reduce the data for representing silhouettes.

Compared to an octree, the proposed method has the following advantages. First, the visual hull tree can represent an object's details well, even if it does not descend to the leaf node level. Second, we can decode the visual hull tree using a similarity transformation without any post-processing.

The remainder of this paper is organized as follows. Section II reviews studies related to our method, and we describe our proposed visual hull tree in Section III. Experimental results are presented in Section IV. Finally, we conclude the paper in Section V.

## II. RELATED WORKS

### A. OCTREE

An octree is considered to be a three-dimensional algorithm of a quad-tree representation and is introduced in [1]. To generate an octree, voxel data are divided into octants repeatedly, representing voxel data hierarchically. The octree reduces the cost of voxel data in terms of memory space, and there are many algorithms based on the octree representation. For example, a modeling scheme using the octree is proposed in [10]. Furthermore, many studies involving rendering, encoding, and deep learning have been developed [3], [11], [12]. To further reduce the memory space associated with an octree, a directed acyclic graph (DAG) is used, and is generated by merging the same pattern of nodes as those in an octree [2].

### B. VISUAL HULL

The visual hull is a 3-D entity that is generated by the shape-from-silhouettes (SfS) technique. To generate a visual hull, silhouettes are back-projected into 3-D space, and the intersection of the back-projected volumes is calculated [6] (Figure 1a). It is a simple algorithm but requires many iterations to calculate the intersection of volumes. To improve the computation time, 3-D ray-based methods have been proposed [4], [5]. In [5], 3-D rays are projected into the silhouettes, and the intersections of rays and silhouettes are calculated. In this method, the intersection points in 2-D images are also back-projected into the 3-D space using 2-D homography. A second method [4] uses affine rectification so that the rays are projected in parallel onto 2-D images. Because the projected rays are parallel, it is easy to locate the intersections between rays and silhouettes.

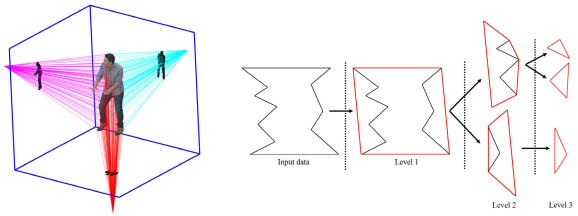



**FIGURE 1.** (a) Shape-from-silhouettes concept. (b) Concavity tree concept.

### C. CONCAVITY TREE

The concavity tree is a data structure that considers concave regions in a 2-D binary image [9] (Figure 1b). To generate a concavity tree, concave regions($\mathbf{C}$) of an object are estimated using a convex hull. Then, the concave regions are estimated again from $\mathbf{C}$, and a hierarchical data structure is formed based on the concave regions. Because the convexity of an object is characterized by a concavity tree, the tree has many applications. For example, shape retrieval and classification can be accomplished by observing the convexity of objects [8], [13]. Encoding objects in 2-D binary images is another application of the concavity tree [14].

## III. VISUAL HULL TREE

The system for generating the visual hull tree is shown in Figure 2. First, we project input voxels to the image planes and generate silhouettes. After acquiring the silhouettes, a visual hull is generated and residuals are found by calculating the differences between a visual hull and the input data. The residuals, then, are represented by silhouettes again. This process is repeated until all residuals are calculated, and then the visual hull tree is generated. Through the visual hull tree, input voxels are modeled by a set of silhouettes and run-lengths. Algorithm details are provided in the subsections that follow.

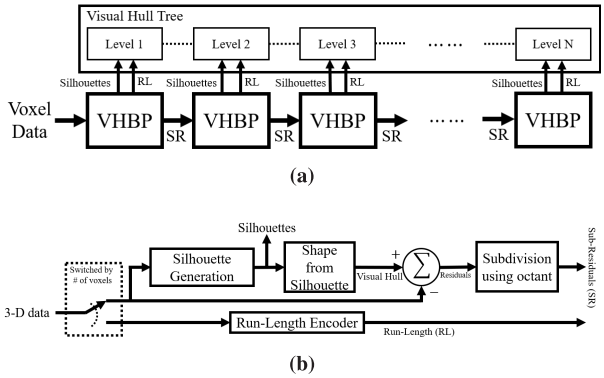



**FIGURE 2.** (a) Overall visual hull tree generator. (b) Visual hull-based predictor (VHBP).

### A. VISUAL HULL COMPUTATION

In this section, we introduce how to generate silhouettes from input voxel data and predict input voxel data using a visual hull. A visual hull is a 3-D entity that is an approximation of

a 3-D object, and the shape-from-silhouettes (SfS) technique is utilized to compute it. The SfS back-projects the regions on the silhouettes into 3-D space and calculates their intersections, generating a visual hull based on these intersections. To model voxels using visual hulls, we generate silhouettes from the voxel data. To generate silhouettes from the voxels, we define virtual cameras around the data and project the voxels onto 2-D image planes. A virtual camera is commonly modeled by projective cameras that are used for modeling real cameras. However, in terms of our method, projective cameras not only increase the complexity in calculating a visual hull, but also make it necessary to encode the camera parameters. To address this, we define virtual cameras as affine cameras. An affine camera is defined as follows.

$$P_A = \begin{bmatrix} M_{2\times3} & t_{2\times1} \\ 0^T_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & t_1 \\ m_{21} & m_{22} & m_{23} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

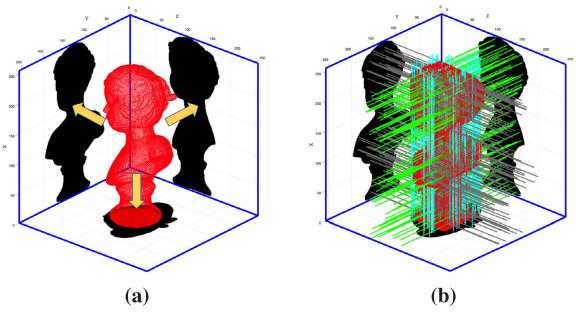
Because the last row of an affine camera is $(0, 0, 0, 1)^T$, an image is generated through the orthogonal projection of 3-D data. As shown in Figure 3, in the case of an affine camera, calculations for silhouettes and a visual hull are simpler than those when using a projective camera. In our work, we set three affine cameras with each camera's principal axis set along the X-axis, Y-axis, or Z-axis; hence a visual hull is easier to calculate. From the particular case of the affine camera, we define the affine cameras as

$$P_{AX} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$P_{AY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$P_{AZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

Through the affine cameras (equations (2), (3), (4)), the silhouettes and a visual hull are generated by analyzing the coordinates of the voxels. In terms of generating silhouettes, voxels are projected into the cameras by multiplying the camera matrices and the coordinates of the voxels, $S_X = P_{AX}V, S_Y = P_{AY}V, S_Z = P_{AZ}V$. Here, $V$ is a homogeneous coordinate of voxels and $S_X$, $S_Y$, and $S_Z$ are homogeneous coordinates of pixels on silhouettes that are generated by $P_{AX}$, $P_{AZ}$, and $P_{AZ}$, respectively. The projection is repeated until all voxels are projected into the affine camera's image planes. However, because equations (2), (3), and (4) are special cases of affine cameras, we can compute the silhouettes by removing specific coordinates. For instance, in terms of equation (2), the Y and Z coordinates of a voxel become the x and y coordinates of an image plane.



(a) (b)

**FIGURE 3.** (a) Silhouette generation and (b) shape obtained from silhouettes using affine cameras.

To compute a visual hull using silhouettes, we may use a ray carving method [4], [5]. A ray carving method defines rays in 3-D space regularly and projects them into the image planes. Then, it calculates the intersection between the projected rays and the contours of silhouettes, and the intersection points are back-projected into 3-D space, from which a visual hull is calculated. This algorithm is a fast method and useful for our system. However, we can compute a visual hull in simpler fashion than ray carving methods by using affine cameras instead. Before computing a visual hull, regions in silhouettes are back-projected into 3-D space as follows:

$$X(\lambda) = P^{\dagger}x + \lambda C \quad (5)$$

where $P^{\dagger}$ is the pseudo inverse matrix of camera matrix $P$, and $x$ is the homogeneous coordinate of a pixel. $C$ is a null space of $P$, and $\lambda$ is a real value. All pixels in silhouettes are back-projected by equation (5), and 3-D space is filled with back-projected voxels. Here, because we use simple affine cameras, back-projected voxels are generated by filling the space along the x-axis, y-axis, and z-axis. For instance, back-projection of a pixel $(x, y)$ onto $S_Z$ is simply represented by $(x, y, k)$, where $0 \leq k \leq N$, and $N$ is the voxel resolution. After back-projecting pixels, a visual hull is calculated by finding the intersections of the back-projected pixels in 3-D space as follows.

$$V = X_X(\lambda_X) \cap X_Y(\lambda_Y) \cap X_Z(\lambda_Z) \quad (6)$$

To calculate equation (6), a bit-wise AND operation is utilized in 3-D space [15].

### B. RESIDUAL COMPUTATION AND MEASUREMENT

Because a visual hull is only an approximation of an object, a visual hull and an object have differences. Thus, to represent voxels using a visual hull, it is necessary to find the residual of a visual hull by calculating the differences between a visual hull and the voxels. To find a residual, we use the exclusive-OR operation applied to a visual hull and the input voxels, thus finding the differences between them [15]. The residuals are composed of a set of 3-D voxel blobs, and each residual is then represented by a visual hull again. If residuals have few numbers of voxels, it would be inefficient to represent them as a visual hull, because many bits can be used for representing the residuals as silhouettes. For this reason, we represent the residuals as 3-D run-lengths if the number of voxels in residuals is less than a specified threshold. The above process is repeated until all residuals disappear, and a visual hull tree that consists of silhouettes and 3-D run-lengths is

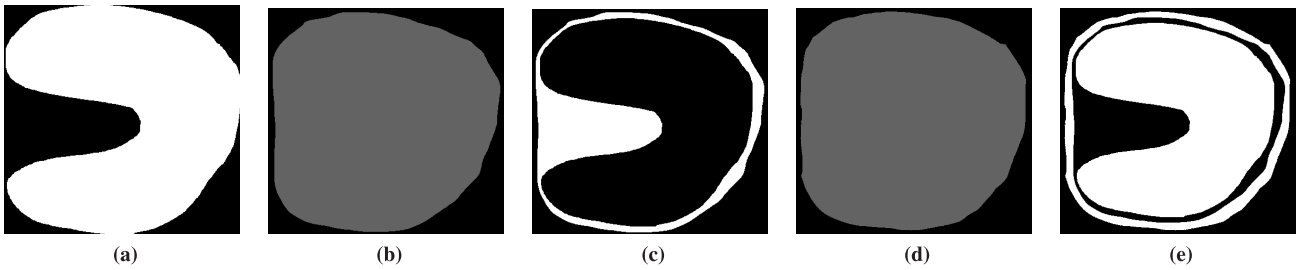



**FIGURE 4.** Example of divergence of the residuals in a 2-D image. (a) Input data. (b) Visual hull of (a). (c) A residual of (a) and (b). (d) Visual hull of (c). (e) A residual of (c) and (d). From the observation of (c) and (e), the residuals show similar shapes and are not convergent.

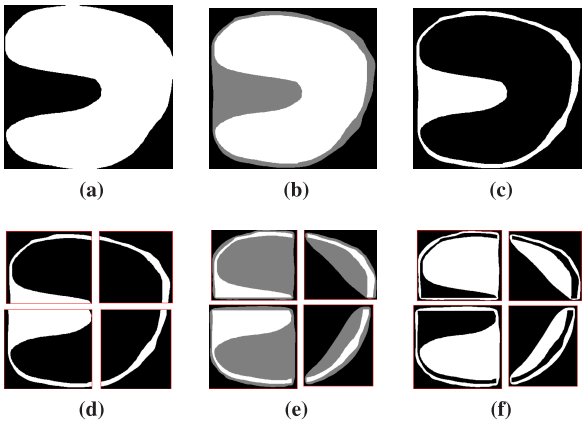



**FIGURE 5.** Example of the computation of residuals in 2-D space. (a) An input. (b) The input (white region) and a visual hull (gray region). (c) Residuals of the input and the visual hull. (d) Divided residuals (sub-residuals) and (e) corresponding visual hulls (gray region). (f) Residuals of visual hulls in (e).

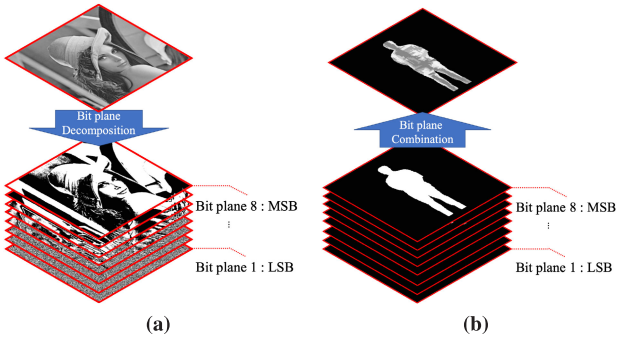



**FIGURE 6.** (a) Concept of bit plane decomposition. (b) Bit plane-based silhouette representation.

**Algorithm 1** Generation of a Visual Hull Tree

**Input:** Input Voxel Data $\mathbb{V}$
**Output:** Visual Hull Tree $\mathbb{VHT}$
  $V_P \leftarrow \mathbb{V}$
  **while** !isempty($V_P$) **do**
    $nb \leftarrow$ Get_The_Number_of_Blob($V_P$)
    **for** $k = 1$ to $nb$ **do**
      $vht, V_{diff} \leftarrow$ Generate_Level_of_VHT($V_P(k)$)
      $V_N \leftarrow$ append_to($V_{diff}$)
      $\mathbb{VHT} \leftarrow$ append_to(vht)
    **end for**
    $V_P \leftarrow V_N$
  **end while**

generated. However, the convergence of the visual hull tree is not guaranteed, because a visual hull usually encloses an object. Figure 4 shows the problem of convergence in 2-D space. In Figure 4-(c) and (e), we note that the residuals have similar shapes repeatedly, and therefore the visual hull tree may not be convergent. To allow convergence of the visual hull tree, the residual is partitioned by subdividing it into octants, which we refer to as sub-residuals. Subdividing takes place after finding the residuals, and each divided residual is represented by a visual hull (Figure 5). Because the subdividing process reduces the size of the residuals, the visual hull tree becomes convergent. Algorithms 1 and 2 describe the overall generation process for visual hull trees.

### C. BIT PLANE-BASED SILHOUETTES REPRESENTATION

The visual hull tree consists of silhouettes and 3-D run-lengths. In terms of silhouettes, various methods can be utilized to represent silhouettes, such as 2-D run-lengths, a chain code, or a quad-tree. However, depending on the geometry of the 3-D object, many silhouettes may be required. Hence, a large amount of data may be required if traditional binary representations are utilized for encoding silhouettes. To handle this issue, we propose the bit plane-based representation, which merges silhouettes into a gray image. As we represent silhouettes using gray images, we can utilize lossless gray-scale image encoders, which have been studied extensively.

A bit plane of a gray-scale image is a binary image that is composed of a set of bits corresponding to a given bit position based on intensity values. As shown in Figure 6a, bit plane 8 indicates the most significant bits based on their intensity values, and bit plane 1 contains the bits with least significant intensity values. In this manner, we consider silhouettes in a visual hull tree as bit planes. The simplest way of gathering silhouettes is to consider each silhouette as a bit plane. However, this approach generates many bit planes; thus, it requires much data. To reduce the number of bit planes, we gather silhouettes into a single bit plane without loss of the information describing the silhouettes. To achieve

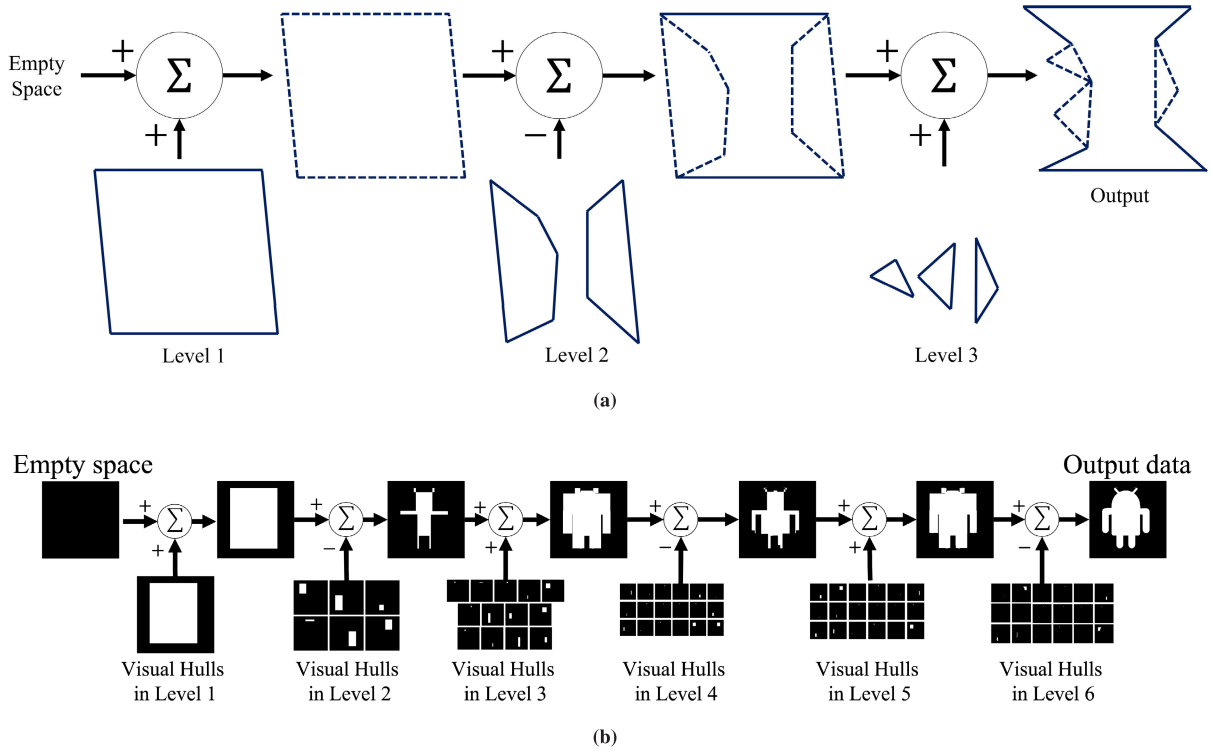



**FIGURE 7.** Example of decoding process. (a) shows the decoding process of a concavity tree. (b) shows the decoding process of a visual hull tree in 2-D space. Both decoding processes are similar.

**Algorithm 2** Generate Level of VHT

**Input:** Input Voxel Data $\mathbb{V}$
**Output:** Visual Hull Tree $\mathbb{VHT}$, Difference between a visual hull and an object $V_D$

```
nv ← The_Number_of_Voxel(𝕍)
if threshold ≤ nv then
  V_S ← Split_by_Octant(𝕍)
  for k = 1 to size(V_S) do
    S ← Generate_Silhouettes(V_S(k))
    VH ← Generate_Visual_Hull(S)
    V_D ← XOR(VH,𝕍)
    𝕍𝕀𝕋.sil ← append_to(S)
  end for
else
  𝕍𝕀𝕋.run ← append_to(𝕍)
end if
```

**Algorithm 3** Bit Plane-Based Silhouettes Representation

**Input:** Visual Hull Tree $\mathbb{VHT}$
**Output:** Image $\mathbb{I}$

```
for k = 1 to max_Level do
  num_sil ← 𝕍𝕀𝕋(k).num_sil
  for j = 1 to num_sil do
    tmp_I ← 𝕍𝕀𝕋(k).sil(j)
    for i = 1 to num_sil do
      if !is_overlapped(tmp_I, 𝕍𝕀𝕋(k).sil(i)) then
        tmp_I ← Union(tmp_I, 𝕍𝕀𝕋(k).sil(i))
      end if
    end for
  end for
  Update_Bit_Plane(𝕀, tmp_I)
end for
```

this, we project blobs of silhouettes onto a single bit plane while determining whether they are overlapped.

To decode the visual hull represented by bit plane-based silhouettes, we establish some rules for bit plane-based representation.

1) Each bit plane contains silhouettes present only at the same level in a visual hull tree. If silhouettes at the same level are represented by more than one bit plane, the number of bit plane is recorded in memory space.
2) Each set of silhouettes on the XY, YZ, and ZX planes is represented by the bit plane-based method. As a consequence, there are three sets of images used to represent silhouettes.
3) If the number of bit plane is more than 8, an additional image is created.

The detailed process is described in algorithm 3.

### D. DECODING OF VISUAL HULL TREE

In Figure 7, we describe the example of decoding process in 2-D space. The decoding of a visual hull tree is similar to the decoding of a concavity tree. Figure 7a shows an example of the decoding process in a concavity tree. The decoding process involves repeated subtraction and addition

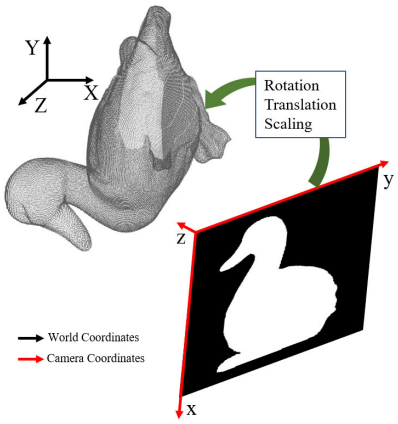



**FIGURE 8.** Relationship between the camera coordinate and the world coordinate. From this relationship, a visual hull tree can be decoded using similarity transformation.

of concave regions. In a similar manner to a concavity tree, the decoding process of a visual hull tree starts from the root node, and subtraction or addition is performed on the visual hulls (Figure 7b).

$$D_{t+1} = \begin{cases} VH_t + D_t & \text{if } t \text{ is an odd number} \\ \overline{VH_t} \cdot D_t + VH_t \cdot \overline{D_t} & \text{if } t \text{ is an even number} \end{cases} \quad (7)$$

Here, $D_t$ is a decoded voxels in level $t$. $VH_t$ is a visual hull in level $t$. To compute $VH_t$, silhouettes in level $t$ undergo SfS that is introduced in subsection III-A. If level $t$ is odd, a union (OR operation) for addition is used; otherwise, an exclusive OR (XOR) operation for subtraction is utilized. Specifically, in the first level, a visual hull is generated by SfS. In the second level, visual hulls are also generated by SfS, and an XOR operation is performed on the decoded voxels in the first level and the visual hull in the second level. Here, the residuals in the second level also undergo an XOR operation. In the third level, an OR operation is performed between the decoded voxels in the second level and the visual hull in the third level. Through the OR operation, the merged visual hulls become new visual hulls because the OR operation makes any output a 1, except for the case when the inputs are (0,0). From the above description, the visual hulls and 3-D run-lengths in the odd levels undergo the OR operation. In the even levels, the XOR operation is applied to the visual hulls and 3-D run-lengths. After reaching the final nodes, the visual hull tree is wholly decoded, and the voxel data is generated. Algorithm 4 presents the decoding process for a visual hull tree.

### *E. ROTATION, TRANSLATION, AND SCALING OF VISUAL HULL TREE*

In the decoding stage of a visual hull tree, the computation of a visual hull is essential. To compute a visual hull, we set simple camera matrices that are mentioned in the previous section. Specifically, we set a camera's extrinsic matrices as a $4 \times 4$ identity matrix that represents an orthogonal projection. The extrinsic matrix defines the relationship between a camera coordinate and world coordinate, which consists of rotation, translation, and scaling (Figure 8). If the extrinsic matrix is an identity matrix, the camera and world coordinates are the same. Otherwise, the camera and world coordinates

**Algorithm 4** Decoding of a Visual Hull Tree

**Input:** Visual Hull Tree $\mathbb{VHT}$, Max Level of visual hull tree $L$
**Output:** Decoded Voxel Data $\mathbb{V}$

```
V ← Generate_Empty_Space(Resolution)
for k = 1 to L do
  S ← Get_Silhouettes(VHT,L)
  R ← Get_Run_Length(VHT,L)
  if size(S)!=0 then
    for j = 1 to size(S) do
      VH ← Compute_Visual_Hull(S(j))
      if mod(k,2)==1 then
        V ← Union(VH,V)
      else
        V ← XOR(VH,V)
      end if
    end for
  end if
  if size(R)!=0 then
    for j = 1 to size(R) do
      if mod(k,2)==1 then
        V ← Union(R,V)
      else
        V ← XOR(R,V)
      end if
    end for
  end if
end for
```

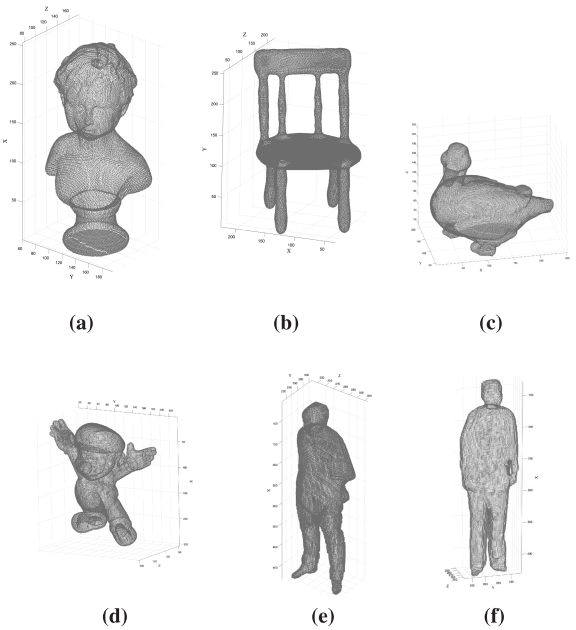



**FIGURE 9.** Datasets used to evaluate the proposed method. (a) 'Bust,' (b) 'Chair,' (c) 'Duck,' (d) 'Mario,' (e) 'VCL Man1,' and (f) 'VCL Man2.'

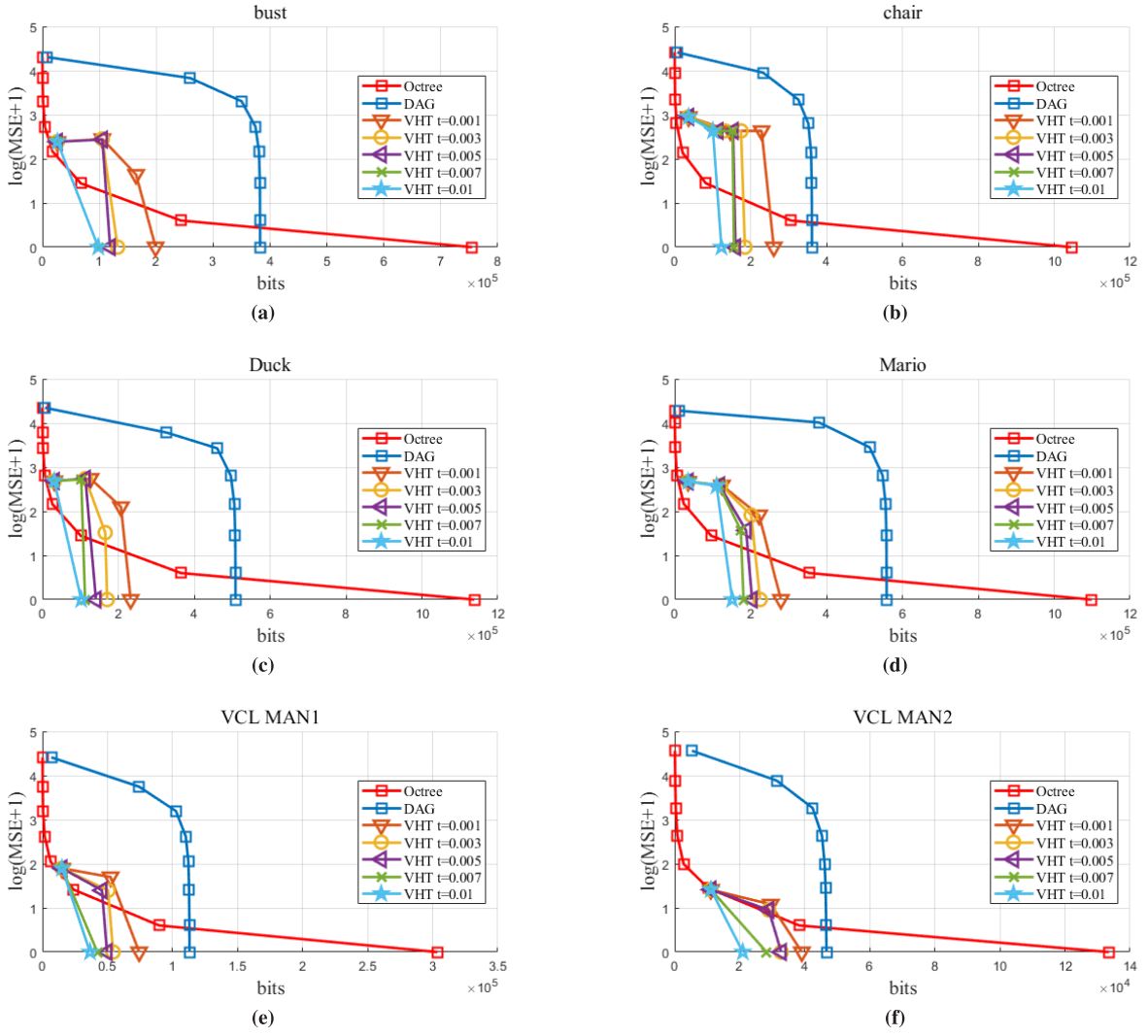



**FIGURE 10.** RD-curves based on number of levels. (a)-(f): 'Bust,' 'Chair,' 'Duck,' 'Mario,' 'VCL Man1,' and 'VCL Man2.'

have a relationship under a similarity transformation. From the above analysis, we can directly decode a similarity-transformed object if we modify the extrinsic matrices with suitable parameters, such as rotation angle, translation, and scaling. Equation (8), as shown at the bottom of the page, indicates the relationship between a camera matrix and an extrinsic matrix $R_Z R_Y R_X T$. In equation (8), $c_x$ and $s_x$ are respectively the cosine and sine of $\theta_x$, while $S_X$, $S_Y$, and $S_Z$ are scaling factors along the $X$-axis, $Y$-axis, and $Z$-axis, respectively. $t_X$, $t_Y$, and $t_Z$ are translation factors. Before computing a visual hull, camera matrices are modified using equation (8), and then visual hulls are computed. In terms of residuals, only the start points and end points of run-lengths are transformed using equation (8). Through the above process, we can decode transformed voxels without post-processing.

## IV. SIMULATION RESULTS

In simulations, we experimented with six different voxel data images: 'Bust' [16], 'Chair' [16], 'Duck' [17], 'Mario' [17], 'VCL Man1' [4], and 'VCL Man2' [4]. 'Bust,' 'Chair,' 'Duck,' and 'Mario' are mesh data and we voxelize them

$$
\begin{aligned}
P_X &= P_{AX} R_Z R_Y R_X T \\
&= P_{AX} \begin{bmatrix} c_z & -s_z & 0 & 0 \\ s_z & c_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_y & 0 & s_y & 0 \\ 0 & 1 & 0 & 1 \\ -s_y & 0 & c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_x & -s_x & 0 \\ 0 & s_x & c_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_X & 0 & 0 & t_X \\ 0 & S_Y & 0 & t_Y \\ 0 & 0 & S_Z & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{8}
$$

in 3-D space (Figure 9). 'VCL Man1' and 'VCL Man2' are multiple-view image data, thereby their geometry is reconstructed using the ray carving method [4], [5] (Figure 9). We set the resolution of all voxel data as $256 \times 256 \times 256$.

To model voxel data as a visual hull tree, a parameter (the threshold) should be set. The performance of our method therefore varies depending on the threshold. In the experiments, we set a threshold by multiplying the number of voxels and $t$. Here, $t$ is a positive number less than 1.0. In the experiments, we varied $t$ in the range of 0.001 to 0.01 and constructed visual hull trees. Additionally, we compared our method to octree. To evaluate the validity of the proposed method, we use a geometric measure that is used for MPEG Point Cloud Compression [18]. It proposes assessment criteria regarding the quality of the geometry and colors of a point cloud; we use only the geometric criterion for evaluations in the experiments. The geometric criterion is defined as follows:

$$d(V_o, V_{deg}) = max(d_{rms}(V_o, V_{deg}), d_{rms}(V_{deg}, V_o)) \quad (9)$$

where, $V_o$ is the original voxel or point cloud data and $V_{deg}$ is the degraded voxel or point cloud data. $d_{rms}$ is defined as

$$d_{rms}(V_o, V_{deg}) = \sqrt{\frac{1}{K} \sum_{v_o \in V_o} \|v_o - v_{dnn}\|^2} \quad (10)$$

Here, $K$ is the number of voxels in $V_o$. $v_o$ is an element of $V_o$, and $v_{dnn}$ is the nearest neighborhood of $v_o$. In our experiments, to observe the various scales of errors, we use the logarithm of equation (9). To handle $d(V_o, V_{deg}) = 0$, we add 1 to equation (9) and take its logarithm.

$$d_f = log(d_{rms}(V_o, V_{deg}) + 1) \quad (11)$$

Equation (11) is calculated for each level of the visual hull tree and octree, observing the geometric error while traversing the leaf nodes of the trees.

Figure 10 presents the geometric errors regarding bits. The silhouettes of a visual hull tree are merged into gray images by bit plane-based representation. In this experiment, we represent the gray images using the png file format, which is a lossless image encoder. To find curves in Figure 10, we decode the visual hull trees along with the level of detail, and then equation (11) is calculated. Furthermore, the numbers of bits are measured cumulatively by adding the bits for each level of the visual hull tree. To compare the proposed method with previous methods, we evaluate octree, which is a popular representation method for point clouds, and DAG, which is a modified version of octree. To evaluate the previous methods, equation (11) is also calculated using decoded voxel data along with the level of detail, and the numbers of bits for each level are measured in the same way as for the visual hull tree. In Figure 10, the level of the visual hull tree becomes shallow as the threshold approaches 0.01. Furthermore, from observing Figure 10, the visual hull tree converges at a lower level than the octree. Because the test voxel data have $256 \times 256 \times 256$ resolution, the octree always has eight levels. However, the depth of the visual hull tree varies depending on the threshold and the shape of voxel data.

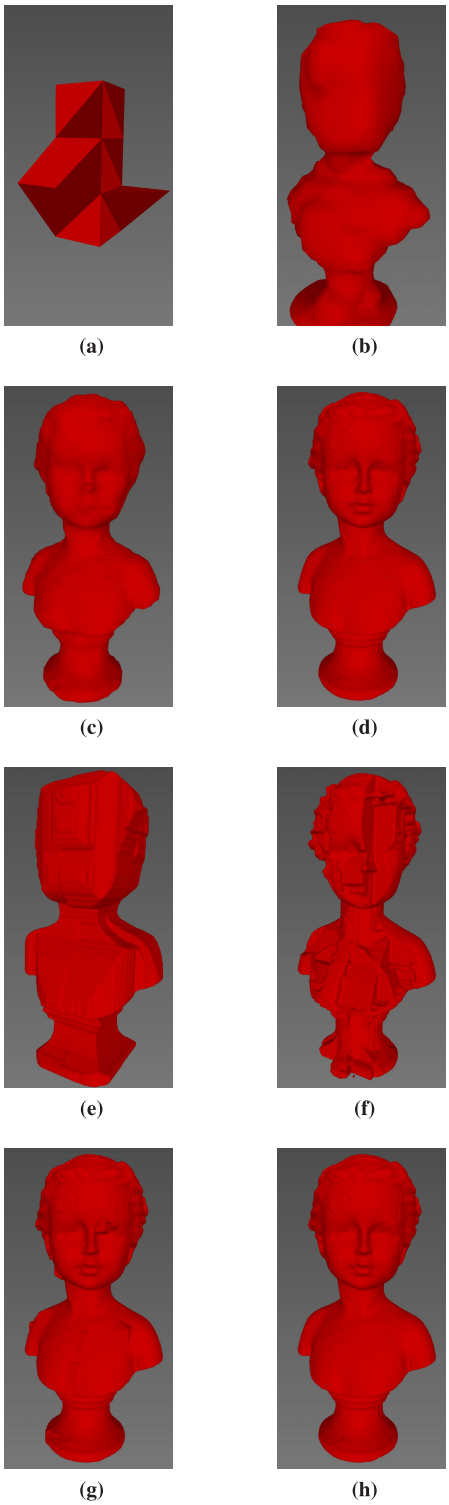



**FIGURE 11.** Subjective quality of 'bust' represented by octree. (a)-(d) show levels 2, 4, 6, and 8 of an octree. (e) to (h), level 1 to level 4 of a visual hull tree. In the visual hull tree, we set the threshold to $t = 0.001$.

We can also observe the number of bits used to represent a visual hull tree, octree, and DAG in Figure 10. From the figure, the visual hull tree uses fewer bits to represent 3-D

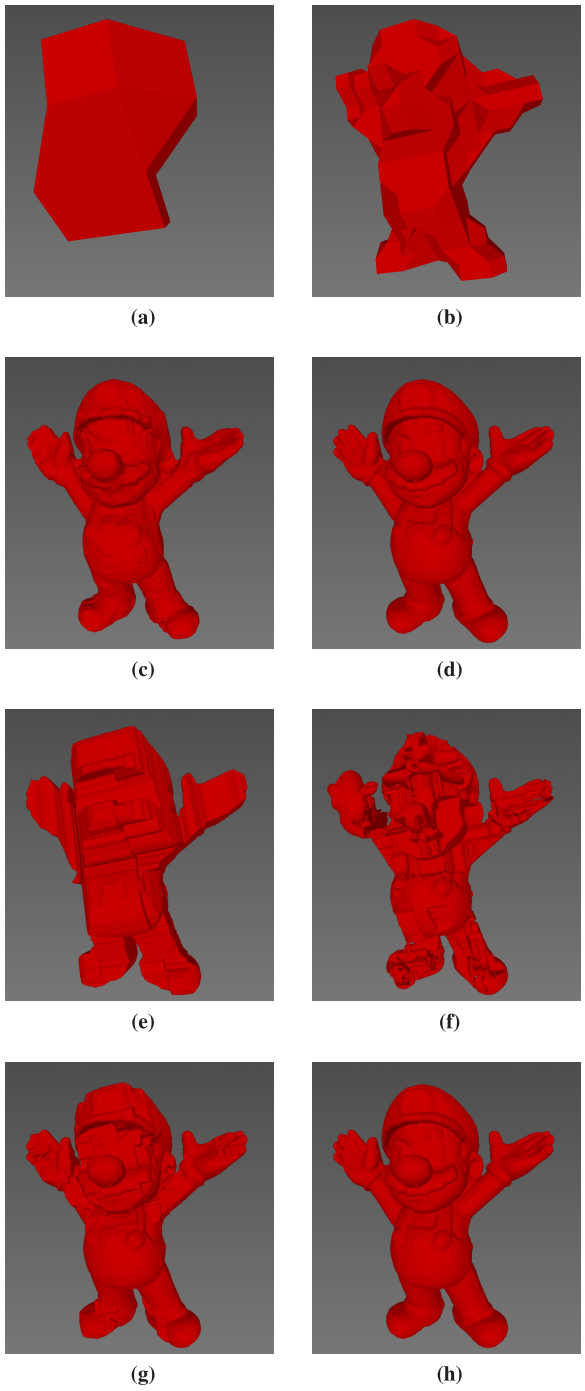



**FIGURE 12.** Subjective quality of 'Mario' represented by octree. (a)-(d) show levels 2, 4, 6, and 8 of an octree. (e) to (h): level 1 to level 4 of a visual hull tree. In the visual hull tree, we set the threshold to $t = 0.001$.

data compared to octree and DAG. We also present subjective quality images in Figures 11 and 12. From Figures 11 and 12, the visual hull tree not only converges quickly, but also achieves similar geometry at all levels.

## V. CONCLUSION

In this paper, we propose a new progressive representation method, a visual hull tree, for 3-D voxel data. A visual hull is a 3-D entity that is generated by SfS technique. SfS computes a visual hull using silhouettes, which means that a visual hull is represented by silhouettes that have a small amount of data. To take advantage of it, we model voxel data as visual hulls. Because the visual hull does not describe concave regions, we estimate concave regions by calculating the differences between voxel data and visual hulls. The concave regions are also represented by visual hulls, and this process is repeated until all concave regions are processed. Finally, the input voxel data are represented by hierarchical visual hulls, i.e., a visual hull tree. Although the visual hull tree does not traverse leaf nodes, it produces voxel data similar to the input data. Hence, the visual hull tree is useful for progressive coding. Also, the visual hull tree is invariant for the similarity transform because we can modify the cameras, which are used for computing visual hulls. Moreover, to reduce the data of the visual hull tree, we propose the bit plane-based silhouettes representation. From the experimental results, the proposed method demonstrated superior performance to the octree algorithm.

As future work, we plan to develop lossy coding strategies for a visual hull tree. Furthermore, we expect to apply these strategies to visual hull trees, because the proposed method was motivated by concavity trees.

## REFERENCES

[1] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *Comput. Graph. Image Process.*, vol. 14, no. 3, pp. 249–270, Nov. 1980.
[2] V. Kämpe, E. Sintorn, and U. Assarsson, "High resolution sparse Voxel dags," *ACM Trans. Graph.*, vol. 32, no. 4, p. 101, 2013.
[3] S. Laine and T. Karras, "Efficient sparse voxel octrees," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 8, pp. 1048–1059, Aug. 2011.
[4] S. S. Hwang, H.-D. Kim, T. Y. Jang, J. Yoo, S. Kim, K. Paeng, and S. D. Kim, "Image-based object reconstruction using run-length representation," *Signal Process., Image Commun.*, vol. 51, pp. 1–12, Feb. 2017.
[5] S. Kim, H.-D. Kim, W.-J. Kim, and S.-D. Kim, "Fast computation of a visual hull," in *Proc. Asian Conf. Comput. Vis.* Berlin, Germany: Springer, 2010, pp. 1–10.
[6] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 2, pp. 150–162, 1994.
[7] M. Attene, M. Mortara, M. Spagnuolo, and B. Falcidieno, "Hierarchical convex approximation of 3D shapes for fast region selection," in *Computer Graphics Forum*, vol. 27, no. 5. Hoboken, NJ, USA: Wiley, 2008, pp. 1323–1332.
[8] O. El Badawy and M. Kamel, "Shape retrieval using concavity trees," in *Proc. 17th Int. Conf. Pattern Recognit. (ICPR)*, vol. 3. Piscataway, NJ, USA: IEEE, Aug. 2004, pp. 111–114.
[9] J. Sklansky, "Measuring concavity on a rectangular mosaic," *IEEE Trans. Comput.*, vol. C-21, no. 12, pp. 1355–1364, Dec. 1972.
[10] D. Meagher, "Geometric modeling using octree encoding," *Comput. Graph. Image Process.*, vol. 19, no. 2, pp. 129–147, Jun. 1982.
[11] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 4, pp. 828–842, Apr. 2017.
[12] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3577–3586.
[13] V. Cantoni, A. Ferone, A. Petrosino, and G. S. di Baja, "A supervised approach to 3D structural classification of proteins," in *Proc. Int. Conf. Image Anal. Process.* Berlin, Germany: Springer, 2013, pp. 326–335.
[14] J. Bajon, M. Cattoen, and S. D. Kim, "A concavity characterization method for digital objects," *Signal Process.*, vol. 9, no. 3, pp. 151–161, Oct. 1985.
[15] W.-J. Kim, S.-D. Kim, and H. Radha, "3D binary morphological operations using run-length representation," *Signal Process., Image Commun.*, vol. 23, no. 6, pp. 442–450, Jul. 2008.

[16] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3D mesh segmentation," *ACM Trans. Graph.*, vol. 28, no. 3, p. 73, 2009.
[17] A. Nouri, C. Charrier, and O. Lézoray. (2017). *Greyc 3D Colored Mesh Database, Technical Report*. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01441721
[18] R. Mekuria, Z. Li, C. Tulvan, and P. Chou, *Evaluation Criteria for PCC (Point Cloud Compression)*, ISO/IEC Standard JTC1/SC29/WG11, Feb. 2016.

**TAE YOUNG JANG** (Member, IEEE) received the B.S. degree in electrical engineering from Sejong University, Seoul, South Korea, in 2013, and the M.S. degree from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2015, where he is currently pursuing the Ph.D. degree in electrical engineering. His research interests include image-based 3D modeling, 3D data compression, and augmented reality.

**SEONG DAE KIM** (Life Senior Member, IEEE) received the B.S. degree in electronics engineering from Seoul National University, Seoul, South Korea, in 1977, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science, Seoul, in 1979, and the Dr.-Ing. degree in electrical engineering from ENSEEIHT, INPT, Toulouse, France, in 1983. He has been a Professor with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, since 1984. His research interests include 3-D reconstruction, computer vision, pattern recognition, image coding, and image processing.

**SUNG SOO HWANG** (Member, IEEE) received the B.S. degree in electrical engineering and computer science from Handong Global University, Pohang, South Korea, in 2008, and the M.S. and Ph.D. degrees from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2010 and 2015, respectively. His research interests include image-based 3D modeling, 3D data compression, augmented reality, and simultaneous localization and mapping systems.