IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 23, 2021, accepted August 9, 2021, date of publication August 16, 2021, date of current version August 24, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3105417

A Real-Time Sculpting and Terrain Generation System for Interactive Content Creation

CHIEN-WEN CHEN[1], MIN-CHUN HU[2], (Member, IEEE), WEI-TA CHU[3], (Senior Member, IEEE), AND JUN-CHENG CHEN[4], (Member, IEEE)

[1]Graduate Program of Multimedia Systems and Intelligent Computing, National Cheng Kung University and Academia Sinica, Tainan 701, Taiwan
[2]Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan
[3]Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan
[4]Research Center for Information Technology Innovation, Academia Sinica, Taipei 115, Taiwan

Corresponding author: Min-Chun Hu (anitahu@cs.nthu.edu.tw)

This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant 109-2218-E-006-019.

ABSTRACT Volumetric representation is widely used in digital sculpting and terrain generation due to its highly modifiable characteristic. However, storing the entire raw voxel map requires a lot of memory, which limits its application on general machines. In this work, we propose to use a novel data structure based on the concept of layered depth-normal images to replace the commonly used scalar field for storing the volumetric information. The proposed data structure can be quickly edited, and the isosurface can be extracted by using general methods such as Marching Cubes and Dual Contouring. The proposed sculpting system can edit models with multiple resolution levels, and the surface between models of different resolutions can be connected seamlessly. Our method can achieve similar results as the existing volumetric methods while significantly reduces the memory usage and computation time. The model created by our system can be used in common game engines to make highly interactive games. Furthermore, all of our processes are parallel-friendly. We implemented a virtual reality digital content creation tool based on the proposed method to demonstrate the effectiveness and feasibility of our method.

INDEX TERMS Real-time sculpting, terrain generation, virtual reality, volumetric sculpting.

I. INTRODUCTION

Three-dimensional models are widely used in games, industrial design, and film production. The emerging demands for 3D models makes it important to develop a friendly modeling tool to create a large amount of materials. In this work, we aim to design a modeling application which can be easily used by non-professional user to express their creativity. The 3D modeling process can be divided into the following categories: parametric modeling, non-uniform rational B spline (NURBS) modeling, polygonal modeling, polygonal sculpting, and volumetric sculpting. Parametric modeling, NURBS modeling, and polygonal modeling methods require highly professional ability to create models and are not friendly for beginners. Sculpting based methods provide a more intuitive model editing process which allows the user to draw a model based on the shape of the brush without knowing the basis of the model (i.e., the vertices, edges, and triangles). This work targets at developing a modeling software which can be intuitively used in real-time applications, and therefore we choose to use the modeling method based on volumetric sculpting. We investigate the problems of volumetric sculpting method and propose the corresponding solutions.

For traditional modeling software, the user edits the model through the 2D screen, keyboard, and mouse. It is difficult to control the brush in a 3D virtual space through the 2D screen. The user needs to constantly change his/her view to edit various areas of the model, which makes it difficult for beginners to create even a simple prototype of 3D model. With the rise of virtual reality (VR), there are many applications proposed for artistic creation in a virtual space [1]–[7]. These VR applications allow the user to directly "draw" 3D content in the 3D environment. The success and rapid popularity of Oculus Medium [6] and Kodon [7] prove that 3D modeling based on the volumetric method is efficient enough for VR interaction, and the flexibility of editing allows the user to freely draw in any area in the 3D space. Moreover,

The associate editor coordinating the review of this manuscript and approving it for publication was Wen-Sheng Zhao.

the VR environment is suitable for multi-person cooperation and we have developed a collaborative VR sculpting prototype based on conventional scalar field based method [8]. Users enjoy co-creating with others in the VR environment; however, we encountered the problem of large memory usage when drawing a large scale model. In this work, we propose a memory efficient method for real-time sculpting, which allows the user to create larger-scale models.

The flexibility of editing is an attractive feature to general user. Also aiming at allowing flexible editing, Minecraft [9] uses low-resolution blocks and various materials to enable the user to construct the scene easily with the charming of voxels. ASTRONEER [10] uses simple intuitive editing operations to achieve highly destructible terrain, which shows the potential of volumetric based applications. In this work, in addition to developing a sculpting application, we also hope that the created content can be exported and imported to another application for further usage, such as being taken as part of destructible objects in another game to interact with the player. Therefore, we have to consider the compatibility to make the system easy to be integrated with the existing game engines (e.g., Unity 3D [11] and Unreal Engine 4 [12]).

In this work, we develop a volumetric based real-time VR sculpting and terrain generation system, which allows the user to flexibly edit the 3D content. The main contributions of our system are summarized as follows:

- We propose a memory efficient data structure for volumetric sculpting based on the concept of layered depth-normal images (LDNIs) [13].
- Based on the proposed data structure, We design commonly used sculpting tools including drawing, erasing, smoothing, and deformation.
- A seamless multi-resolution mechanism is proposed to efficiently represent models with different resolutions.
- A GPU-aided implementation pipeline is introduced for real-time volumetric sculpting. The overall pipeline of the proposed method is shown in Fig. 1.

The rest of this manuscript is organized as follows. We first review the related researches in Section II. The proposed data structure is introduced in Section III. The details of model manipulation and mesh extraction methods are described in Section IV, Section V, and SectionVI. The implementation details and the experimental results are discussed in Section VII and Section VIII, followed by the conclusion and future work in Section IX.

II. RELATED WORK

The 3D modeling process can be divided into the following categories: parametric modeling, non-uniform rational B spline (NURBS) modeling, polygonal modeling, polygonal sculpting, and volumetric sculpting. We first introduce commonly used modeling methods and software.

- **Parametric modeling** method is commonly used in computer-aided design (CAD) software such as SOLIDWORKS [14] and Autodesk Inventor [15]. A 3D model is generated according to the parameters set by the user (e.g., the thickness or radius), and these parameters strictly restrict the appearance of the model. Since each instruction/operation is affected by the previous ones, the parametric design requires careful planning and highly professional ability. The advantage of parametric modeling is that the intermediate instructions/operations can be easily reused and composed to create similar models through simple parameter adjustment.
- **NURBS modeling** method [16] uses mathematical formulas to accurately represent the geometry of objects, and it is suitable for modeling curved objects. NURBS modeling software (e.g., Rhino [17]) is often used in industries that require high surface quality of the designed model, such as automobiles and aerospace.
- **Polygonal modeling** method [18] directly operates on the vertices of the mesh. Famous 3D modeling software such as Maya [19] and Cinema 4d [20] use this kind of modeling technique. Compared with parametric modeling and NURBS modeling methods, directly operating on the model provides a more intuitive editing way for the user. Since all objects in polygonal modeling are composed of polygons (e.g., triangles or quadrilaterals), there is no real sphere or arc, which needs to be approximated by multiple continuous polygons. Although the surface of the model is not as smooth as that in parametric modeling or NURBS modeling, polygonal modeling is more efficient in terms of rendering. The polygonal models are mostly used in the industry of movies, animations, and games.
- **Sculpting based modeling** methods provide a more intuitive model editing process than polygonal modeling. The user can draw a model based on the shape of the brush without knowing the basis of the model (i.e., the vertices, edges, and triangles). The sculpting based modeling methods can be classified into polygonal sculpting and volumetric sculpting. In a polygonal sculpting software (e.g., Zbrush [21]), the user directly operates on the vertices and surfaces of the model.

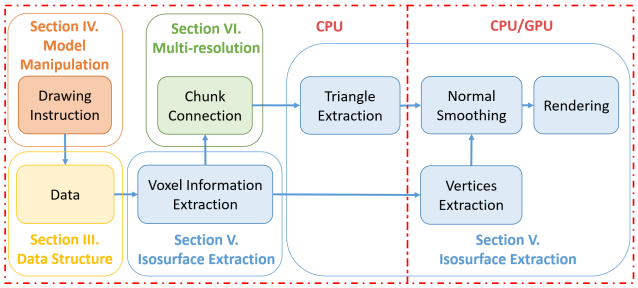

FIGURE 1. The overall pipeline of the proposed rel-time volumetric sculpting method. When the user manipulates the model, the system first edits the stored data according to the drawing instruction. Then the voxel information is extracted from the data and a multi-resolution mechanism is applied in order to extract seamless triangle mesh. After obtaining the vertices and triangles, the normal vector of each vertex is calculated and then used for rendering the model. The steps in the left red box are computed on CPU and the steps in the right red box can be computed on CPU or on GPU.

For example, the user can edit the model by moving the positions of the vertices, adding new vertices, or deleting existing vertices. The advantage of directly operating on the model is that the user can build a precise model by assigning the vertices to specific positions. However, it is more complicated to calculate the new topology after adding/deleting a vertex. The user needs to plan ahead with the base mesh. Moreover, the user cannot extrude endlessly or hollow out the mesh. Therefore, polygonal sculpting is more suitable for hard bodied surfaces, like armour and buildings. In a volumetric sculpting software (e.g., 3D-Coat [22]), the system divides the space into voxel grids and then stores the model data in these small grids like pixels in the image. The user operates on the implicit data (e.g., the scalar field) instead of the mesh, and the system directly renders based on the implicit data or extracts the mesh by a specific method (e.g., Marching Cubes [23]) first for rendering. Because each voxel is independent and does not interfere with each other, volumetric sculpting is more flexible than polygonal sculpting. For volumetric sculpting, the current state of the model does not affect the following editing process. Moreover, it is easy to merge different objects for the user. However, the fineness of the model is limited by the resolution of voxel. Volumetric sculpting is suitable for creating quick concept or organic stuff, which does not require sharp features. Since our target is to develop a modeling software which can be intuitively used in real-time application, we choose to use the volumetric sculpting based method, investigate the problems of this modeling method, and propose the corresponding solutions.

In addition to develop a sculpting application, we also aim at allowing the user to intuitively create interactive objects (especially terrain) for game development. Here, we introduce commonly used approaches of terrain generation. According to the requirement of different applications, terrain generation can be implemented with different approaches. Due to the characteristic of the terrain, two-dimensional heightfield is the most common data structure used for storing and rendering. Two-dimensional heightfield is suitable for real-time editing and rendering of large-scale terrain because we only store and operate on 2D data. Moreover, the topology of the grids is fixed, which means the results can be calculated quickly with efficient memory usage. This technique is adopted by the terrain system in popular game engines such as Unreal Engine 4 and Unity 3D. However, the heightfield cannot represent terrain structures with multiple vertical layers such as overhanging cliffs, caves, or arches. Feature-based methods are commonly used in the application of rapidly automatic procedural generation of large-scale terrain. The system automatically generates terrain according to rough user input such as simple outlines of the terrain, material type of the specified terrain [24], [25], or other external forces such as the extrusion of plates [26]. It would be useful for game developer to create terrain in the developing stage, but the player cannot modify the details of the terrain while playing. Compared with heightfield and feature based methods that are suitable for editing terrain in the game developing stage, volumetric methods have the advantage of allowing players to edit the terrain during the playing stage.

The concept of volumetric sculpting was first introduced by Galyean *et al*. [27]. The 3D space is divided into voxel grids, each voxel is marked as inside or outside the solid region, and then Marching Cubes algorithm is used to extract the isosurface based on the state of the cell (eight neighboring voxels construct a 3D cell). Each vertex of the mesh is located at the midpoint of one cell edge. With the advancement of computing power, many volumetric sculpting methods based on scalar field have been proposed [28]–[30]. The value of the scalar field represents the distance from voxel to the surface of the solid region (A positive value indicates that a sample lies in the empty space, and a negative value indicates that a sample lies in the solid space). We can use these values to extract more precise vertex positions and keep more details. However, a critical problem of the scalar field methods is the inefficiency of memory usage. Therefore, researchers are committed to exploring a method that can effectively save memory and simultaneously achieve acceptable data access time. Since the values of the scalar field are important only around the surface of the model, we can store only the narrow-band data (also known as truncated signed distance field (TSDF)). For example, we only store the value between -1 and 1. The voxel values outside the model and far from the surface are all set to 1, while the values inside the model and far from the surface are all set to -1. In most cases, if the model is not too broken, the scalar field values of -1 (as well as 1) appear continuously as a cluster in the voxel space, and therefore the model representation can be compressed by different methods such as octree [31], hierarchical run-length encoding [32], and dynamic tubular grids [33]. However, these methods would have additional computation burden of decompressing data if the user want to access or modify the model. Another way is to only store the voxel chunks around the surface without compression. With the spatial hashing technique [34], [35], we can store data densely, which is beneficial for streaming between CPU and GPU. However, the size of a voxel chunk is fixed and part of the memory space is not used, resulting in more memory waste than just storing the narrow-band data. Volumetric Dynamic B+ tree (VDB) [36] provides dynamic scalar field data compression and access functions. The B+ Tree based structure can support constant-time random access operations like lookup, insertion, and deletion. The overall goal of VDB is to consume only as much memory as is required to represent active voxels, that is, only narrow-band data are stored. In Section VIII, we will show that the memory usage for storing narrow-band data is still about twice than our method. With the rise of deep learning techniques, there have been many works compressing the volumetric data through neural networks [37], [38], and these methods can achieve high compression rate. However, these methods are lossy

compression methods, and the reconstructed results would contain noises. In addition, the computation process of neural network inference is time consuming and not suitable for the real-time sculpting application.

A volumetric based terrain generation method named Transvoxel was introduced in [39], which provides the theoretical basis and implementation details of a complete and practical real-time voxel-based terrain rendering system. Well-known modeling tools such as Oculus Medium and Voxel Plugin [40] for Unreal Engine 4 are developed based on this method. They use three-dimensional scalar field to store the volumetric information, and Marching Cubes algorithm is applied to extract the isosurface. In order to develop the LOD system, they introduce an algorithm named Transition Cells [41] to patch the seams, cracks, and holes between chunks of different resolutions by inserting an additional spatial structure. They further introduced methods for applying texture maps and advanced shading techniques to voxel-based terrain meshes. In this work, to significantly reduce the memory usage, our data structure is designed based on the concept of LDNIs instead of using the scalar field data. Moreover, we propose a new method of the multi-resolution mechanism, which does not require inserting additional structures between chunks and can be easily applied to achieve the LOD system.

LDNIs is a modeling method proposed for Computer aided design (CAD). Compared with scalar field which stores 3D voxel data, the LDNIs method stores the intersection points and normal vectors of the model in images which are orthogonal in three-axis. The number of images required for each axis is the maximum number of intersections between the grid lines in that direction and the model. This method can be beneficial for storing data in GPU memory. Since there are not too many intersections between a grid line and the model in general, the required memory can be reduced greatly. In order to keep sharp features, the LDNIs method adopts Dual Contouring algorithm [42] to extract the isosurface of the model. The required input data of Dual Contouring algorithm are the intersection positions and the normal vectors of the model and the grids, which are exactly the values stored in LDNIs. Based on the data structure used in LDNIs, commonly used operations in CAD (e.g., boolean operations and offsetting) have been implemented. Moreover, boolean operations between complex static models can be computed efficiently base on GPU. Inspired by LDNIs, in this work we store the model by the intersections and the normal vectors, which represent the model with less memory compared to scalar field data. We further propose a suitable data structure, several editing tools, and a multi-resolution mechanism to improve the real-time sculpting performance.

III. DATA STRUCTURE

A two-dimensional example of a model intersecting with the grid lines in the $x - y$ plane is shown in Fig. 2. We connect the intersections of the model and the grid lines (indicated by gray and yellow points) to represent the shape of the model. Two points form a segment which cover a solid region on a grid line. In order to facilitate storage and calculation on GPU, the LDNIs method stores the intersections in images orthogonal to x, y, or z axis. The number of images required for each axis is the maximum number of intersections between the grid lines in that direction and the model. As shown in Fig.3, since the maximum number of intersections between a sphere model and a grid line in the x direction is two, the LDNIs method stores the data in two images (Fig. 3(c) and (d)). However, this kind of data structure results in some unused memory space (the white points in Fig. 3(c) and (d)) and does not support random insertion or deletion. Furthermore, for a real-time sculpting application in which the models are usually edited frequently by the user, the number of intersection points changes all the time. If we store the data as images, it not only wastes a lot of memory usage but also needs to reallocate data frequently when the user edits the model. To develop a real-time VR sculpting application in which rendering requires a lot of GPU memory and computing power, GPU resources are even more precious. Compared to GPU memory, the main memory space is much larger and allows us to edit the model at higher resolutions. In addition, connecting chunks of different resolutions requires lots of logical operations, which are more suitable to be calculated on CPU than on GPU. Since when performing undo/redo functions, the memory arrangement can be managed more efficiently in the main memory, we choose to store the data in the main memory and design a data structure to deal with operations for real-time sculpting.

For each axis, we store segments on a grid line as a list (as shown in Fig. 2). Each segment represents a solid region on the grid line and consists of two points. These two points represent the start and end of the segment, which are the intersections of the model and the grid lines. The normal vectors of the surface at the intersections can be also stored in the list to extract the isosurface. It is worth noting that points on the same grid line have the same two coordinate values (e.g., the same x and y), and therefore we only need to store the value in the rest direction (e.g., the z value). Segments in a list are sorted according to the stored values. As illustrated in Fig. 2, we can observe that for a manifold model, the number of intersections between the model and each grid line should be even, and each intersection pair represents the two endpoints of a solid section. Therefore, we can use a set of segments S to represent a manifold model. Fig. 3(b) shows the 3D illustration of the proposed data structure. The white lines indicate the grid lines in the x direction, and a segment list is used for storing the intersections on each grid line.

Shown in (a) into the proposed representation and LDNIs in the x axis.

IV. MODEL MANIPULATION

In this section, we first introduce how to perform drawing and erasing mesh based on the segment list data representation (Section IV-A) and show how to edit data with different shapes of brushes (Section IV-B). Section IV-C describes the

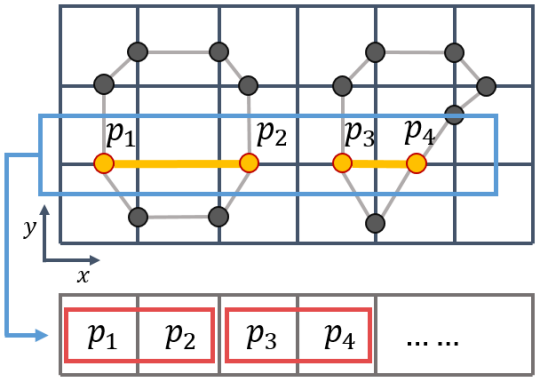

FIGURE 2. Two-dimensional example of the proposed data structure in the $x-y$ plane. The gray and yellow points indicate the intersections of the model and the grid lines. The yellow points are the endpoints of the segments on a grid line in the x direction. The yellow lines indicate the region covered by the segments which also represent the region inside the model. For each grid line, a list is used to store the segment endpoints and these endpoints are sorted by the stored values. Two adjacent values in the list represent a segment (indicated by red box).

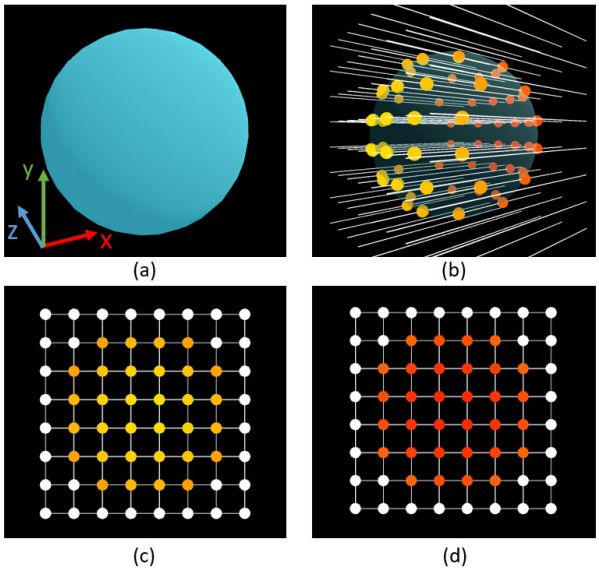

FIGURE 3. Illustration of the difference between the proposed representation and the LDNIs representation.(a) A sphere model. (b) For our proposed representation, each white line indicates a grid line in the x direction and the points indicate the intersections between the model and the grid lines. For each grid line, we use a list to store the positions of the intersections. On the other hand, LDNIs representation uses images to store the intersection information. (c) and (d) are LDNIs representation of the sphere model in (a). The colors of the points (from yellow to red) represents the normalized position in the x axis (from 0 to 1), and the white points are unused memory space.

implementation details of some special tools (e.g., smoothing and deformation) developed based on the proposed data structure. We also implement the color and material system based on the proposed method (Section IV-D).

A. SEGMENT MANIPULATION

Updating the segment set S of the whole 3D scene can be decomposed to updating segment lists $\{l_i\}$. A segment list l_i represents the unit spatial information in one direction (x, y, or z) and is composed of several segments. Assume there are n segments $s_0, \ldots, s_n$ (i.e., $2n$ points) in l_i, two successive points construct either a solid section or an empty section. When drawing or erasing a new segment s to l_i, we need to update the list based on the current section states and the new segment state (solid for drawing and empty for erasing). Algorithm 1 illustrates the updating method. To be more specific, we observe the states of sections where the start and end points of the new segment (s_s, s_e) are located. If the state is different from the new segment, the point is inserted into the list, and if state is the same, no action is required. Finally, we delete all the points between (s_s, s_e) in the list. The model we draw generally does not have too many points in a list; therefore, computation for drawing or erasing can be very fast. We discuss the computation performance in detail in Section VIII.

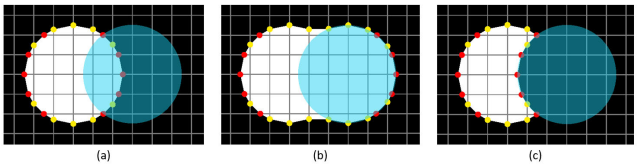

FIGURE 4. (a) A brush (indicated by a blue circle) is used to edit an existing model. (b) The result of drawing. (c) The result of erasing.

B. DRAWING WITH BRUSH

The manifold model can be represented by a segment set, and the result of performing constructive solid geometry (CSG) operation based on two manifold is also a manifold. Any manifold model can be taken as a brush, and CSG operation can be performed directly on the segment-based data. For each CSG operation, the brush model is converted into a segment set B that operates on S (the segment set of the current model). Here we use $(\oplus, \ominus)$ to represent brush operations. For each segment $b_0, \ldots, b_n$ in B, we perform Algorithm 1 based on corresponding segment list in S. Each b_i is set as solid when applying $S \oplus B$, and is set as empty when applying $S \ominus B$. In practice, any manifold brush can be used to edit the model by calculating the intersections of the brush and the grids. The example results of drawing and erasing operations are shown in Fig. 4.

C. SPECIAL TOOLS

Brushes with some special functions are commonly used in digital content creation, such as deformation and smoothing. In order to implement these functions based on the proposed data structure, all operations must be manifold. First, we introduce how to perform smoothing. For each endpoint of the segments inside the brush, we extract the voxel state set V_c and point set P_c of all the cells around it to interpolate the final position of the endpoint. Take Fig. 5 as an example, to calculate the smoothed position of p (indicated by the yellow point), which is stored in a y-directional list, we first extract

Algorithm 1 Segment Manipulation

Input: s_s (the coordinate value of the start point of the input segment), s_e (the coordinate value of the end point of the input segment), $solid_s$ (state of input segment), and l (the segment endpoint list to be edited.)

```
index = 0;
flag = FALSE;
while index ≤ length of l and l[index] < s_s do
    index++;
    flag = !flag;
end
if solid_s ≠ flag then
    Insert s_s to l at index;
end
while index ≤ length of l and l[index] < s_e do
    flag = !flag;
    Remove the value at index from l;
end
if solid_s ≠ flag then
    Insert s_e to l at index;
end
```

its neighboring points on left, right, forward, and backward direction, denoted as n_l, n_r, n_f, n_b, respectively. For example, we examine edges $e1, \ldots, e5$ in order to find the position of the left neighboring point n_l (as shown in Fig. 5(a)). Note that we only extract one neighboring point in each direction and average the positions of the four neighboring points to interpolate the final position p_s of the segment endpoint p with a smoothing strength value w (ranges from 0 to 1), which is formulated as

$$p_s = \frac{(n_l + n_r + n_f + n_b)}{4} \times w + p \times (1 - w). \quad (1)$$

The smoothing strength value w affects how fast the model is smoothed and the user can adjust the value while using our system. Moreover, to achieve reasonable smoothing result, if the extracted neighboring point is on edge e_4, we sample a point on edge e_1 with the corresponding value $y - d$ to be the position of n_l (as shown in Fig. 5(b)). Similarly, if the neighboring point is on edge e_5, we sample a point on edge e_3 with the corresponding value $y+1+d'$ be the position of n_l. However, if we directly move the endpoint of the segment to the calculated position p_s (indicated by the blue point), it will cause a non-manifold result. Therefore, we draw a sphere automatically with a radius of 1/2 displacement at the center between these two points. Fig. 6(a) and (b) show example results of smoothing.

Deformation operations such as extruding or squeezing are more difficult to be converted to manifold operations directly based on the proposed data structure, but we can achieve that in a more intuitive way described as follows. After applying isosurface extraction algorithm introduced in Section V, we can obtain the triangle mesh inside the brush and directly

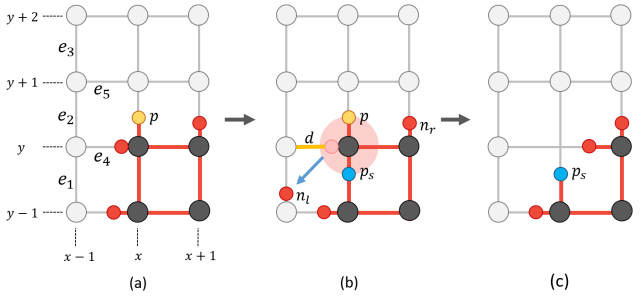

FIGURE 5. The process of the smoothing operation. (a) We use dark gray points to represent the voxels inside the solid area. Red points and lines indicate the segments. Edges $e1, \ldots, e5$ are the sampling order of the neighboring points of p. (b) After obtaining the smoothed position p_s, we draw a sphere with a radius of 1/2 displacement at the center between these two points (indicated by a light red circle) to produce a manifold result. (c) The result of smoothing.

move the vertices. The resulting segment data can be obtained by calculating the intersections of the deformed mesh and the grid lines. Note that in order to prevent self-intersecting of the mesh, we constrain the step size of each vertex displacement to 0.3. Fig. 6(c) and (d) show example results of deformation.

D. DRAWING MATERIAL

As shown in Fig. 7(b), the proposed data structure is sparse. If we only store the material information along with the vertex information, it would not be able to express the material inside the model. For example, when the materials are different at the start and end points of a segment, we do not know the boundary of the two materials. To solve this problem, we store material information in the same way as the volumetric information. Unlike the volumetric segments which need to be stored in three directions, the material segments can be stored in only one direction because there is only one material value for each cell. Each material segment represents a sequence of consecutive voxels with same material. Based on the above design, the user can modify the material or the model individually when drawing. To achieve terrain texture mapping, we adopt the triplanar projection algorithm [43] to obtain the uv value of each vertex from the position and normal value. The user can use multiple materials for one chunk, and the color of adjacent vertices with different materials is interpolated accordingly.

V. ISOSURFACE EXTRACTION

Extracting the isosurface of the model is important for applications such as path generation of computer numerical control (CNC) tool, physical interaction, and parting line generation of mold design. With the ability of extracting isosurface of the model according to the user's modification in real-time, our system can be widely used in general game engines. Well-known volumetric isosurface extraction algorithms such as Marching Cubes, Surface Nets [44], Dual Contouring, and Dual Marching Cubes [45] can be applied to our system. In this section, we introduce how to extract the surface of the model through Marching Cubes and Dual

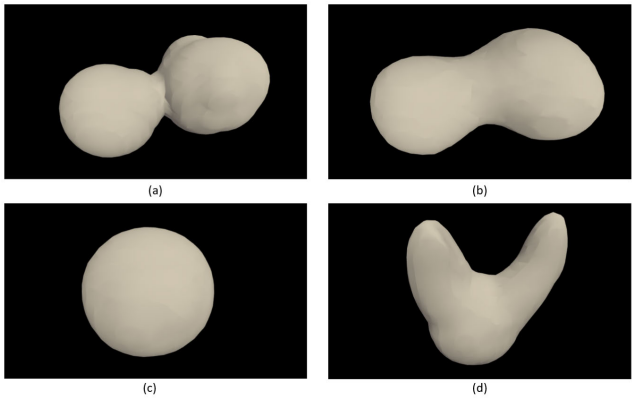

FIGURE 6. (a) The model before smoothing operation. (b) The model after smoothing operation. (c) The model before deformation operation. (d) The model after deformation operation.

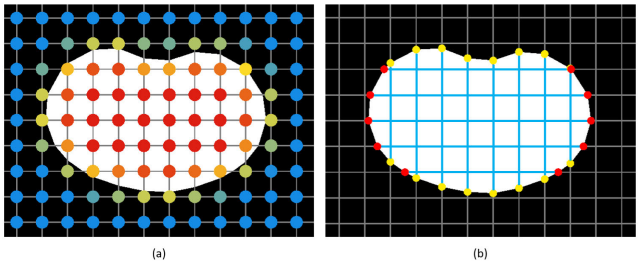

FIGURE 7. Two-dimensional examples of a manifold model described by (a) the scalar field based method and (b) the proposed method. We use points with different colors to represent different signed distance function values in the scalar field based method (from red to blue indicate −1 to 1). For the proposed method, red points indicate the endpoints of segments in x direction and yellow points indicate the endpoints of segments in y direction. The blue lines indicate the region covered by the segments.

Contouring algorithm based on the proposed data structure and the comparison of the meshes extracted from Marching Cubes and Dual Contouring algorithm is shown in Fig. 8.

Marching Cubes algorithm is usually used to extract surfaces from scalar field data. A positive value in scalar field indicates that a sample lies in the empty space, and a negative value indicates that a sample lies in the solid space. Therefore, the voxel state set V (whether each voxel is inside or outside the model) can be easily obtained from the sign of the distance function values. The point set P (the positions of the vertices) of the model surface corresponds to the set of points whose distance function value is zero. As shown in Fig. 7, voxels are points uniformly distributed in the space, and four neighboring voxels construct a 2D cell (eight neighboring voxels construct a 3D cell). Generally speaking, each point of P lies on the edge of the cell, where one endpoint has been classified as lying ''outside'' in empty space and the other endpoint has been classified as lying ''inside'' in solid space. The exact position of the vertex can be calculated by linearly interpolating the two scalar values that are connected by that edge. After obtaining the point set P and the voxel state set V, the Marching Cubes algorithm can be applied to extract the triangle mesh. We examine the voxel states of the eight corners of each 3D cell and generate one or multiple triangles in each cell according to the look-up table [23]. The positions of points in P are the intersections of the model surface and the grids. We use segment set S to represent a manifold model and take the endpoint set in S as P_s. The points in P_s and the corresponding points in P are located on the same cell edge, and therefore we can use P_s to replace P. The voxel state of each voxel can be obtained by examining whether it is covered by a segment. Since the vertex coordinates extracted by Marching Cubes algorithm are limited to the edge of the cell, sharp features are difficult to be retained. Marching Cubes algorithm is more suitable for applications that do not need to keep detailed features, such as natural terrain construction. It only stores the intersection points and does not require normal information, and therefore it reduces lots of memory usage. In addition, it extracts the surface of the model by looking up the table, which makes the calculation fast and even in real-time on a single-threaded CPU.

Unlike the Marching Cube algorithm, the vertices extracted by Dual Contouring algorithm can be anywhere in the cell, so the extracted mesh can present sharp features. For applications requiring accurate model details (e.g., CAD, sculpting), we can extract the isosurface by applying Dual Contouring algorithm. Dual Contouring method requires Hermite data, which are the intersection points between the model and the grids and the normal vectors at intersection points. When a cell has any edge that intersects the model (i.e., the distance function value is zero), it implies there is a vertex in the cell. The vertex position in the cell is obtained by minimizing the error function defined based on all the intersections and the normal vectors in that cell [42]. However, the optimization process is time-consuming, which limits the amount of edited content for CPU-based calculation. Since the vertex in each voxel can be calculated independently, we further design a GPU-aided pipeline as shown in Fig. 1 to enable the user to edit a large region of the model.

We first extract the voxel information of the region that is going to be updated, including whether the voxels are inside the model, the intersections on the cell edges, and the normal vectors. These data are then transferred to GPU memory to calculate the vertex in each cell. Since the triangle information can be extracted without knowing the vertices, we use CPU for triangle extraction and connecting chunks with different resolution levels. After extracting the triangle information, we transfer these data to GPU memory and combine the vertex information into complete mesh. Before rendering, we smooth the normal vector based on the angle between adjacent triangles. Given a triangle T, the normal vector of each vertex on the triangle is calculated by weighted average of the normal vector of the triangle T and the ones of the adjacent triangles $T_a's$. Note that each adjacent triangle T_a shares at least one vertex with the triangle T, and the angle between T_a and T is less than 30 degrees. The weight is calculated based on the area of the triangle. In this way, a vertex shared by multiple triangles might have multiple normal

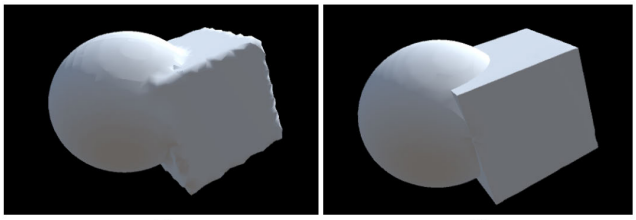

FIGURE 8. The comparison of the meshes extracted from (a) Marching Cubes and (b) Dual Contouring algorithm.

vectors. After rendering the mesh, the triangles, vertices, and smoothed normal vectors are stored in GPU memory to avoid re-calculation.

VI. MULTI-RESOLUTION

Since volumetric isosurface extraction algorithms tend to produce a large number of vertices, multi-resolution mechanism is important to improve rendering performance. We can use fewer vertices to represent flat areas and more vertices to represent details. This could also help us to build the LOD system, which is commonly used in games to improve rendering performance. We divide the 3D space into chunks (a group of cells), each chunk can be edited independently and has its own resolution level. However, there will be gaps between two chunks of different resolutions when using Marching Cubes algorithm to extract mesh. In this section, we introduce how to increase and decrease the resolution based on the proposed data structure and patch the gaps between two chunks without adding any other spatial structure.

A. INCREASING RESOLUTION LEVEL

In order to align the voxel data of chunks with different resolution levels, we half the unit size of the chunk to increase the resolution level of the target chunk, and it consequently increases the number of the segment lists. In an intuitive way, we can obtain the high-resolution information of the segment lists by calculating the intersections of these newly added grid lines and the current model. However, as shown in Fig. 9, for each direction there are three new segment lists per unit cell and two of them locate on the boundary of the cell. According to Marching Cubes algorithm, the edges of triangle also lie on the boundary of the cell. The floating-point error occurs when calculating whether the triangles and the new grid lines intersect with each other. In practice, a more robust, faster and parallel-friendly method is used in our work. We extract the voxel state set V and the point set P_s to determine the new segment information of each cell individually. For the two grid lines lie on the boundary, each one only passes one face of the cell, and we consider this face to design the look-up table for extracting high-resolution information, as shown in Fig. 10. The blue line indicates an edge of the original triangle and the dark gray point indicates the voxel inside the model. We consider the states of the four corners in a cell face, and there are totally $2^4 = 16$ possible distinct cases. For each case, we determine whether the model intersects with the high-resolution grids (indicated by light gray lines) based on the points on the four edges of the cell. If there is an intersection, we "draw"a new segment on the new segment list. Note that this method cannot be applied on the grid line passing through the center of the unit cell. Fortunately, this centered grid line is less likely to locate on the boundary of the triangles obtained by Marching Cubes, and therefore we can directly calculate the intersections of this new grid line and the current model. When the Dual Contouring algorithm is adopted to extract the isosurface, problem that the newly added grid lines are located at the edge of the triangle can be avoided, so the intersection point can be directly calculated to obtain the higher-resolution information.

B. DECREASING RESOLUTION LEVEL

The proposed data structure stores the intersections of the model and the grids. Resolution level only affects how many points to be stored but does not affect the location of these points. Therefore, we can directly discard the high-resolution data when decreasing the resolution of the stored data. The remaining low-resolution segment lists can still form a manifold model. In the LOD system that maintains the high-resolution data, the low-resolution mesh can be extracted from the low-resolution part of the high-resolution segment lists without decreasing the resolution level of the stored data. Conventional scalar field based method has surface shifting problem [39] when triangulating the low-resolution mesh. We need to examine the values at the high-resolution part to construct the low-resolution mesh. Then the low-resolution scalar field values can be directly obtained based on the constructed mesh. In contrast, the proposed data structure only needs to sample the segment lists according to the resolution level, which allows us to build a LOD system in an efficient way. Fig. 11 shows the isosurface extracted from different resolution levels.

C. CONNECTING CHUNKS

As shown in Fig. 12(a), there are gaps between adjacent chunks of different resolution levels when using Marching Cubes algorithm to extract isosurface. Additional mechanisms must be used to patch these gap areas. Fig. 13(a) shows the segment lists on the connecting face, Fig. 13(b) shows the low-resolution information extracted from these segment lists, and Fig. 13(c) shows the high-resolution information extracted from the same segment lists. It can be observed that the high-resolution chunk keeps more details in the extra voxels than the low-resolution chunk, which would produce different meshes and result in gaps. Therefore, we adjust the extra voxels in the high-resolution chunk to produce the same result as the low-resolution chunk on the connecting face, as shown in Fig. 13(d). We change the voxel state of v from empty to solid and add a new point p to construct the edge. The position of p is obtained by calculating the cross point of the edge e in Fig. 13(b) and the grid line. It should be noted that when the state of an extra voxel is changed, we must

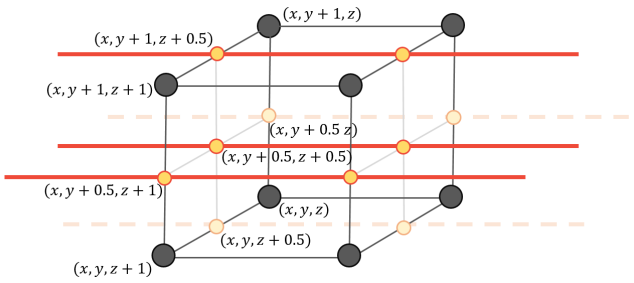

FIGURE 9. The newly added grid lines (indicated by red lines) in a unit cell. Note that two grid lines (indicated by dotted lines) have already been created by the adjacent cell.

ensure that the operation is manifold. That is, if we change the state of a voxel in the x-y plane, we must also add a vertex in the z direction. It is worth noting that these operations only affect the voxel information extracted from the data, i.e., V and P_s, but do not affect the original data. Since we generate each chunk as an independent model during the editing, the number of vertices in adjacent areas need not be the same. However, when exporting the integrated model, we need to split the triangle to connect chunks. The example result of connecting chunks with different resolution levels is shown in Fig. 12(b).

Dual Contouring method naturally supports multi-resolution connection. However, unlike Marching Cubes that generates triangles in each cell independently, the vertices extracted by Dual Contouring are connected to the surrounding cells and the extracted surface of a chunk is related to 27 neighboring chunks ($3 \times 3 \times 3$). These 27 chunks might be in different resolutions, which makes it complicated to implement and calculate the results of chunk connection. Therefore, we take the corner of the chunk as the center to extract the isosurface when applying Dual Contouring algorithm. Fig. 14 illustrates our idea in 2D case, in which a bold square is a chunk. The yellow square shows that if we use a chunk as the center, 8 neighboring chunks are referenced to extract the isosurface. The red square shows that if we use the corner of a chunk as the center, only 4 neighboring chunks are referenced to extract the isosurface. For the 3D case, this idea can reduce the number of referenced chunks to 8 ($2 \times 2 \times 2$).

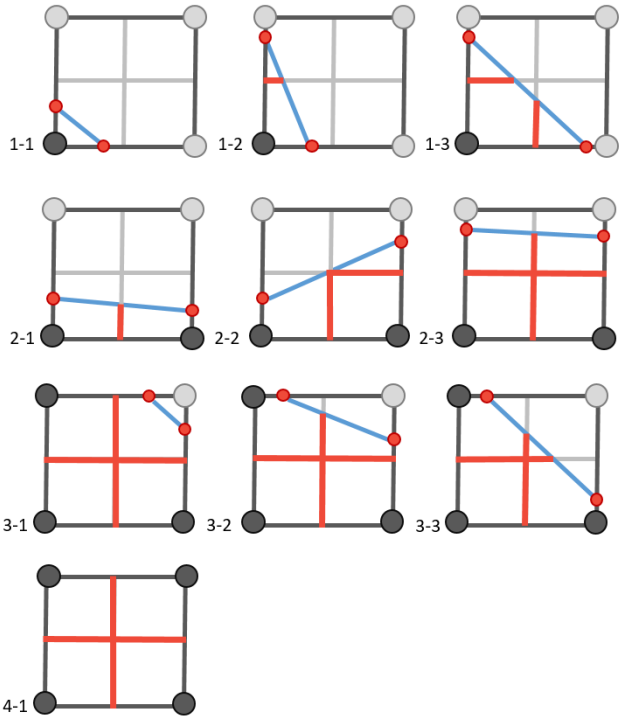

FIGURE 10. A look-up table for extracting high-resolution information. The blue line indicates an edge of the original triangle and the dark gray point indicates the voxel inside the model. Images in the $i-th$ row shows the situations that i voxels are inside the model. A red line indicates a new segment to be added in the high-resolution segment list.

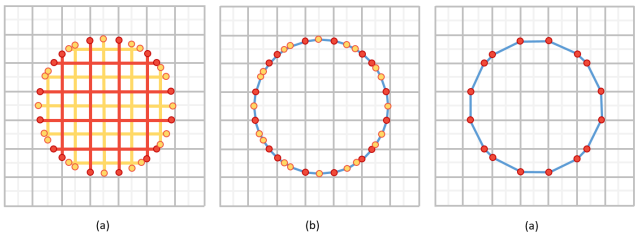

FIGURE 11. (a) The red points and lines indicate the segments on the low-resolution grids, and the yellow points and lines indicate the segments on the extra high-resolution grids. (b) The high-resolution mesh is constructed by the red and yellow points. (c) The low-resolution mesh can be extracted from the low-resolution segment information.

VII. IMPLEMENTATION AND APPLICATION

In order to validate whether the proposed data structure can be widely used, we implemented a VR sculpting and terrain generation application with the Unity 3D Engine. The main algorithm is written in C# language, and HTC VIVE is used to build the VR environment. GPU computing and rendering are respectively implemented in the compute shader and surface shader in Unity 3D.

It is not practical to update the entire mesh map every frame. We divide the space into cell chunks, and each cell chunk is composed of $w \times w \times w$ cell units (in this work w is set as 16). A 2D example of the relation between voxel, cell, segment list, and chunk is illustrated in Fig. 15. Every cross point of the grid lines is a voxel. Each black point with an arrow indicates a segment list storing the intersections of the grid line and the model, and a group of segment lists compose a segment list chunk. In order to extract the information of a cell chunk, $(w + 1) \times (w + 1) \times (w + 1)$ voxel values are needed for the 3D case. For each cell chunk, we extract the volumetric information from the corresponding segment list chunk in x, y, and z direction, respectively. The size of a segment list chunk is $(w + 1) \times (w + 1)$ for the 3D case.

Note that the voxel information is extracted from the corresponding segment list only when the user modifies the voxels in that chunk. When the user edits the model, the system first modifies the corresponding data in the segment list chunks

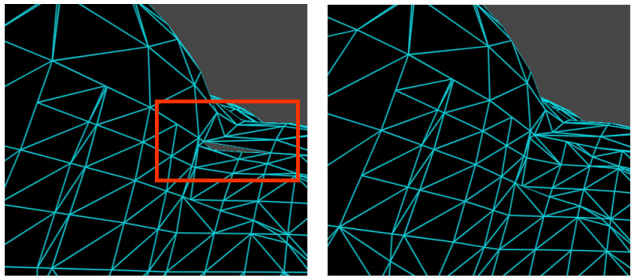

FIGURE 12. (a) The gap between adjacent chunks of different resolution levels when Marching Cubes algorithm is applied to extract the isosurface. (b) The result of applying the proposed patching method.

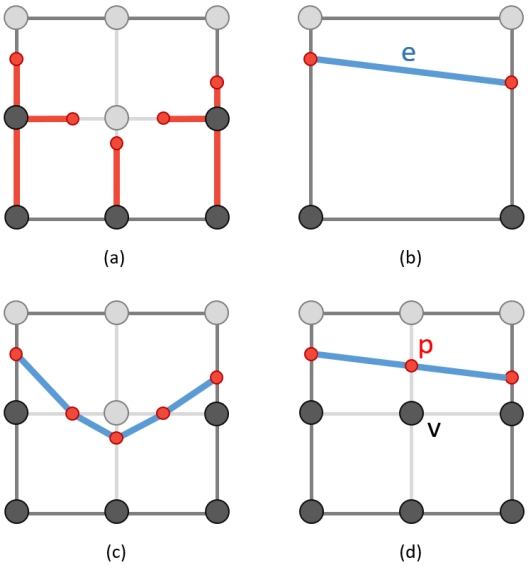

FIGURE 13. (a) The segment lists on the connecting face of two chunks with different resolution levels. (b) The low-resolution information extracted from these segment lists. (c) The high-resolution information extracted from the same segment lists. (d) The extra voxels in the high-resolution chunk is adjusted to produce the same result as the low-resolution chunk in (b). We change the voxel state of *v* from empty to solid and add a new point *p* to construct the edge. The position of *p* is obtained by calculating the cross point of the edge *e* in (b) and the grid line.

and then extracts the volumetric information of the edited chunks (inside the brush range). Isosurface extraction algorithm is applied to extract the triangular surface for rendering. Since the voxel information is required only when updating the mesh of the chunk, the memory used for storing the voxel information can be reused for every chunk. For each chunk, we store the extracted triangles, vertices and smoothed normal vectors in the GPU memory for rendering.

Fig. 16 shows some screenshots of the proposed VR sculpting system and Fig.18(a) shows how the user draws with the HTC VIVE. Through an interface similar to commonly used 2D drawing software, the user can create 3D models in 3D environment based on our VR sculpting system. In addition to the drawing and erasing functions for model editing, we also develop many auxiliary tools for the user, such as line tool,

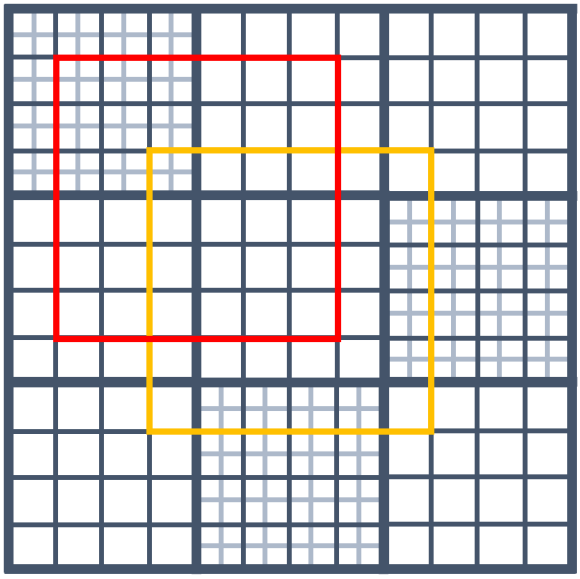

FIGURE 14. Two-dimensional example of referenced regions when Dual Contouring algorithm is applied to extract the isosurface. The yellow square shows that if we use a chunk as the center, 8 neighboring chunks are referenced to extract the isosurface. The red square shows that if we use the corner of a chunk as the center, only 4 neighboring chunks are referenced to extract the isosurface.

alignment tool, layers, undo/redo, model data conversion (converting general model representations into volumetric representation), multi-player mode, saving/reading files, and particle system. These tools are placed on a palette-like user interface attached to the left-hand controller, and the user uses the right-hand controller to choose the tool and edit the 3D model. We also implement shortcut keys on the right-hand controller so that the user can quickly switch to different tools. Moreover, the user can press the buttons on the left-hand controller to move, rotate, and zoom the models in the selected drawing layer. As shown in Fig. 17, the proposed VR sculpting system can be easily used to create an interactive terrain for a game. The produced terrain has the editable characteristics benefiting from the proposed data structure. Moreover, it can be collided with the physics engine in the game engine since we also extract the mesh of the created models. Players can change the terrain through intuitive actions such as launching bombs.

VIII. EXPERIMENTAL RESULTS

We developed the system on a PC with memory size 16GB, Intel i5-9400F CPU, and Nvidia GeForce GTX 1660 GPU. In order to investigate whether the proposed data structure is suitable for real-time sculpting, we compare the memory usage and calculation speed with the scalar field based method. The scalar field and the proposed data structure are both stored in floating-point type. No special tricks are used to speed up. We drew on a canvas with the size of $160\times160\times160$ cells (divided into $10 \times 10 \times 10$ chunks). The brush was set as a sphere with radius 5, 10, 15, 20 units, respectively. We randomly generated 1000 editing instructions of drawing

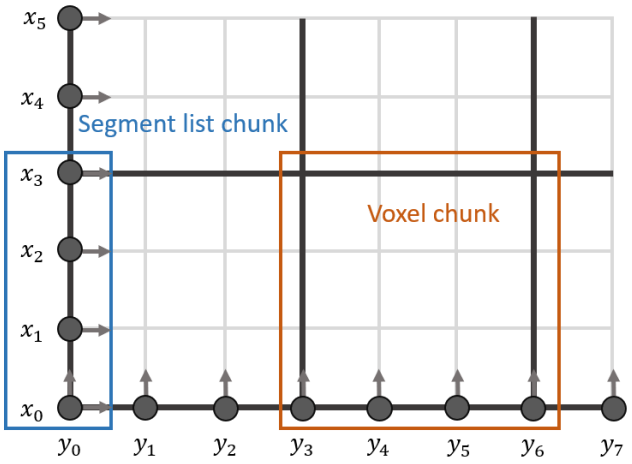

FIGURE 15. A 2D example of the voxel and segment maps with chunk width w of 3 units. Every cross point of grid lines is a voxel, and each black point with an arrow indicates a segment list. The size of the voxel chunk is 4 × 4 (the size is 4 × 4 × 4 in 3D) and the size of the segment list chunk is 4 (the size is 4 × 4 in 3D).

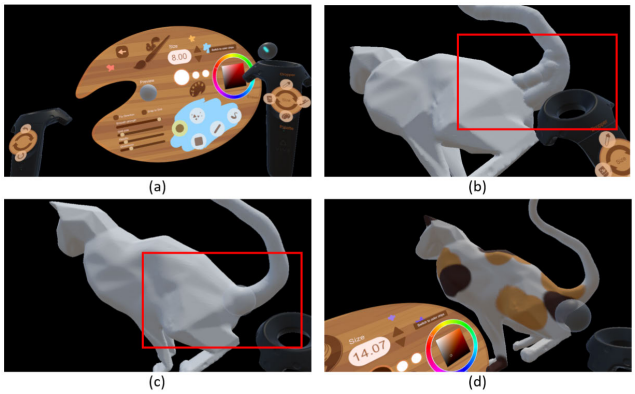

FIGURE 16. (a) The user interface of our VR sculpting application. The user can adjust the parameters of the brush through the VR controllers. (b) The user uses the sphere brush to draw the tail of the cat model. (c) The user uses the smoothing tool to smooth the mesh. (d) The user uses the painting tool to edit the color of the mesh.

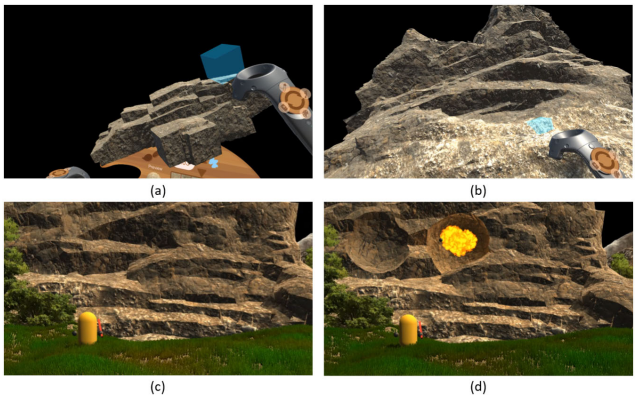

FIGURE 17. (a) The user draws the terrain roughly with the cube brush. (b) The user adjusts the detail of the shape and the materials of the terrain model. (c) The created terrain can be exported for game development. (d) The player launches bombs to destroy the terrain, which can be considered as using the erasing tool.

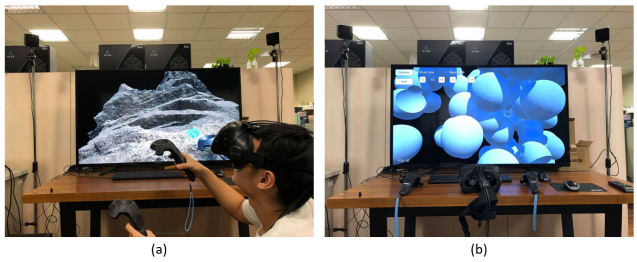

FIGURE 18. (a) The position and rotation of the HTC VIVE controllers and head-mounted display are tracked by the lighthouse. The user can use the controllers to draw and edit the model from different views. (b) The experimental environment for the comparison of computing time and memory usage. We executed random drawing/erasing instructions in the VR environment.

and erasing, which were used to evaluate different methods. For each instruction, we edited the volumetric data according to the brush information and then extracted the mesh for rendering. We list the average computing time to compare the computation performance of our method and the scalar field based method. The memory used by our method increases with the complexity of the model. In contrast, the memory size used by the scalar field is fixed. Therefore, we only list the maximum memory usage for comparison. In addition to conventional scalar field method, we also compare the memory usage with a narrow-band scalar field method.

Compared with conventional scalar field based methods, our method mainly differs in two aspects: (1) the way of editing and storing the volumetric information and (2) the method of extracting the point set P and the voxel state set V. After extracting P and V, the remaining steps are actually the same as the conventional scalar field based methods. Therefore, we compared the time of modifying the volumetric data (denoted as $Time_d$) and the time of extracting the point set and the voxel state set (denoted as $Time_v$) with the scalar field based methods (as shown in Table 1). For the narrow-band scalar method, we applied the run-length encoding (RLE) algorithm to compress the data based on different bandwidths. However, extra calculation time for the RLE algorithm is needed in addition to $Time_d$ and $Time_v$. The extra calculation time for the RLE algorithm (denoted as $Time_{RLE-bandwidth}$) is also listed in Table 1 in terms of different bandwidths. In fact, both our method and the conventional scalar field method can be executed at very high speed through parallel calculation on a high performance PC. However, to further apply the proposed method to other application scenarios (e.g., mobile games) in which the user might use low-end hardware, we evaluate the computing performance on a single-threaded CPU. When editing on the scalar field data, we need to update the signed distance value for each voxel covered by the brush. However, for the proposed data representation, we only need to calculate the intersection points of the brush and the grids, and then add these points to the existing data in a segment manner. That is, the computation complexity of scalar field based methods

TABLE 1. Computation time of the scalar field based method and the proposed method with different brush sizes. We executed 1000 drawing instructions and averaged the execution time of each instruction. The table shows the time of modifying the volumetric data (denoted as $Time_d$), the time of extracting the vertices (denoted as $Time_v$), and the time of applying RLE algorithm (denoted as $Time_{RLE-bandwidth}$).

Data Type and Brush Size	$Time_d$ (ms)	$Time_v$ (ms)	$Time_{RLE-1}$ (ms)	$Time_{RLE-2}$ (ms)	$Time_{RLE-3}$ (ms)
$ScalarField_5$	0.49	1.52	0.74	0.75	0.78
$ScalarField_{10}$	2.02	3.76	1.79	1.92	1.96
$ScalarField_{15}$	5.29	8.27	3.56	3.78	4.33
$ScalarField_{20}$	10.66	14.07	5.59	5.90	6.23
$Segment_5$	0.12	0.26	-	-	-
$Segment_{10}$	0.24	1.57	-	-	-
$Segment_{15}$	0.54	4.27	-	-	-
$Segment_{20}$	0.94	7.68	-	-	-

TABLE 2. The overall calculation time of our method based on different brush size, different isosurface extraction algorithms, and different hardware. The time involves editing the volumetric data, extracting the isosurface, rendering the model, and storing the extracted mesh in each frame. Note that the minimum interval between frames by default is about 11.17ms (90 FPS).

Method	Hardware	$Time_5$ (ms)	$Time_{10}$ (ms)	$Time_{15}$ (ms)	$Time_{20}$ (ms)
Marching Cubes	CPU	11.17	21.76	33.82	52.70
Marching Cubes	CPU+GPU	11.17	12.58	23.25	36.22
Dual Contouring	CPU	22.15	72.18	129.57	187.76
Dual Contouring	CPU+GPU	11.17	23.13	33.72	50.06

is $O(n^3)$ while the proposed method is $O(n^2)$, where n is the brush size. For our method, we calculated the length of each single segment list and found that the maximal length is only 11 (i.e., 22 intersection points) when we drew with a brush with radius of 15. That implies the proposed method can insert/delete a segment to/from the segment list very quickly for an editing operation. When extracting the point set P from the scalar field, the position of a point is calculated by linearly interpolating two scalar values. As mentioned in Section V, we can use the endpoint set P_s to replace P, and we just need to place these points on the edges of the appropriate cell. Therefore, our method has better performance than conventional scalar field based methods in terms of computational efficiency.

Table 2 shows the overall calculation time of our method based on different brush size, different isosurface extraction algorithms, and different hardware. The time involves editing the volumetric data, extracting the isosurface, rendering the model, and storing the extracted mesh in each frame. Since the overall process includes both CPU and GPU calculations, it is difficult for us to accurately calculate the calculation time of the proposed algorithm. Therefore, we compare the frame update speed of the entire system. Since the default maximum frame rate in Unity 3D for VR rendering is 90 FPS (11.17ms), the update time would be 11.17ms even though the computation speed is faster. In general situations, the brush with radius of 5 to 10 is sufficient for the user. For editing with much larger brush size, we can use an additional multi-thread mechanism to allow the user to draw a wide range solid region. Another way to enable real-time editing with large brush size is to avoid extracting triangles every frame. Editing the segment data can be calculated very quickly, but extracting triangles takes most of the time. Hence, we can update the model with an appropriate time interval to achieve real-time large-scale editing experience.

For the conventional scalar field based method, we store the spatial information in 3D dense voxels. Although the chunks can be dynamically allocated in the solid area, the memory size used by each voxel chunk is fixed. When the values of the signed distance function are restricted from $-b$ to b (where b is the width of narrow-band), there will be many consecutive areas with the same value in a large region of solid/empty space, which means that the compression algorithm can be applied to achieve high lossless compression rates. Our method only stores the segment data to represent the solid area of the model, and the memory usage increases with the complexity of the model.

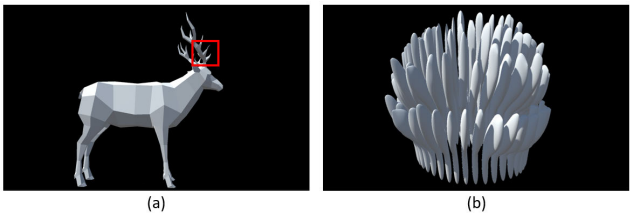

FIGURE 19. (a) A *Deer* model with simpler structure. (b) A *Coral* model with more complex structure.

We use the conventional scalar field method, the narrow-band based method, and the proposed data structure

TABLE 3. We convert a deer model (Fig. 19(a)) and a coral model (Fig. 19(b)) to the proposed representation and the scalar field representation and show the memory usage at different resolutions.

Data Structure	$Deer_{256}$ (byte)	$Deer_{512}$ (byte)	$Deer_{1024}$ (byte)	$Coral_{256}$ (byte)	$Coral_{512}$ (byte)	$Coral_{1024}$ (byte)
Scalar Field	67,108,864	536,870,912	4,294,967,296	67,108,864	536,870,912	4,294,967,296
Narrow-band (b=1)	315,636	1,112,888	3,941,012	8,245,744	39,158,760	169,477,728
Narrow-band (b=2)	601,372	2,221,280	8,056,164	10,289,084	52,448,288	248,835,640
Narrow-band (b=3)	870,828	3,289,472	12,215,088	11,598,492	62,019,756	300,452,144
Ours	121,928	487,248	1,949,936	2,686,236	10,738,520	42,949,188

TABLE 4. The maximum memory usage of the scalar based method, the narrow-band based method, and the proposed method with different brush sizes.

Data Structure	$Memory_5$ (byte)	$Memory_{10}$ (byte)	$Memory_{15}$ (byte)	$Memory_{20}$ (byte)
Scalar Field	16,384,000	16,384,000	16,384,000	16,384,000
Narrow-band (b=1)	1,566,760	3,925,732	3,924,864	3,123,220
Narrow-band (b=2)	2,859,328	7,030,320	7,020,320	5,729,396
Narrow-band (b=3)	3,861,444	9,368,120	9,415,664	7,872,748
Ours	873,416	2,292,240	2,485,520	2,220,744

FIGURE 20. A portion of the *Deer* model (indicated as red box in Fig. 19(a)) reconstructed in the resolution level of (a) 256, (b) 512, and (c) 1024.

to represent two models, i.e., a *Deer* model with simpler structure and a *Coral* model with more complex structures as shown in Fig. 19. Table 3 shows the memory size used for representing each model with different resolutions. Since each grid line would not has too many intersections with the *Deer* model, it requires less memory usage than the *Coral* model. Fig. 20(a) to (c) show a portion of the Deer model (the portion inside the red box in Fig. 19(a)) reconstructed in different resolutions. We further compare the maximum memory usage during the drawing process with the scalar field based method. The memory usage of each method is shown in Table 4. Note that we only calculate the memory usage of the endpoints for the proposed method and the values of the signed distance function for the scalar field based methods, respectively. For the scalar field based method, the amount of memory used by each chunk is fixed regardless of the model. We list the memory usage of the narrow-band based method with the bandwidth b set as 1 to 3. The larger the value of b, the more helpful it is to preserve details and prevent surface shifting when changing the resolution [39], but the more memory it requires. Although only storing the narrow-band value can greatly reduce the memory usage compared to the conventional scalar field method, it results in additional computation burden for maintaining the data structure. Table 3 and Table 4 shows that our method outperforms the conventional scalar field based methods in terms of memory usage.

IX. CONCLUSION AND FUTURE WORK

In this work, we develop a volumetric based real-time VR sculpting system, which allows the user to flexibly edit the 3D content in the 3D environment. The proposed system has high interactivity because the extracted triangle mesh is friendly for physical interaction calculations. It is also suitable for generating terrain and creating destructible objects in game developing. A new data structure is designed based on the concept of LDNIs to reduce memory usage when generating the mesh in a volumetric way. We also introduce a seamless multi-resolution mechanism, which can also achieve the LOD system of terrain rendering. We demonstrate that any manifold model can be easily used as a brush to perform drawing and erasing. However, the proposed method has the following limitations.

- For the narrow-band scalar field based methods, the value stored in each voxel usually ranges from $-b$ to b (b would not be too large), while the value of the segment endpoint is the position of a vertex in a specific direction, which would be relatively large and is prone to floating-point error. Since the precision of a floating-point number is only 6 digits, in practice the data value (including intermediate data in the whole calculation process) is limited to the range from 2^{-8} to 2^{8}. Therefore, if we would like to draw on a large-scale canvas, we need to split the canvas first and use an independent data system with its own coordinates for each sub-canvas.
- Each editing operation in our system should be manifold; otherwise there would be a problem of vertex

missing. For an editing operation that does not directly use a manifold brush, we have to design an additional mechanism to convert it to a manifold operation. For example, to deal with deformation-like operations such as moving, squeezing, and extruding, we directly operate on the vertices of the triangle mesh and then calculate the intersections with the grids. In the worst case, if there is an operation that cannot be converted to a manifold operation, we can still convert the segment data to scalar field data and process the operation by using conventional methods. The segment data can be then obtained from the mesh reconstructed based on the resulting scalar field.

Bolier *et al*. [46] have shown that drawing in the VR space can improve the spatial ability and mental rotation ability of students. However, they draw in the 3D space with 2D brushes just like A-Painter [47] and Tilt Brush. We have done simple user tests on 36 junior high school students and all of them prefer to use 3D brushes than 2D brushes. They also like to draw with their friends in the same virtual environment which can improve the cooperation ability. With the multi-player mode provided by our system, they can draw together even if they are in different physical spaces. Since the proposed method is fast and has high memory efficiency, it has lower hardware requirements and can be more easily promoted to art courses in school. In the future, we are going to investigate whether drawing with 3D brush can be beneficial to improve the spatial ability and mental rotation ability of the students. We will also optimize the user interface to make our system suitable for students, and design a teaching system for art teachers.

REFERENCES

[1] Google. *Tilt Brush*. Accessed: May 10, 2021. [Online]. Available: https://www.tiltbrush.com/

[2] D. F. Keefe, D. A. Feliz, T. Moscovich, D. H. Laidlaw, and J. J. LaViola, "CavePainting: A fully immersive 3D artistic medium and interactive experience," in *Proc. Symp. Interact. 3D Graph. (SI3D)*, 2001, pp. 85–93.

[3] S. Schkolne, M. Pruett, and P. Schröder, *Surface Drawing: Creating Organic 3D Shapes With the Hand and Tangible Tools*. New York, NY, USA: Association for Computing Machinery, 2001, pp. 261–268.

[4] D. Keefe, R. Zeleznik, and D. Laidlaw, "Drawing on air: Input techniques for controlled 3D line illustration," *IEEE Trans. Vis. Comput. Graphics*, vol. 13, no. 5, pp. 1067–1081, Sep. 2007.

[5] B. Maxim and D. Gorgan, "Artworkvr: Novel interaction techniques for virtual painter," in *Proc. 16th Int. Conf. Hum.-Comput. Interact., (RoCHI)*, A. Moldoveanu and A. J. Dix, Eds. Bucharest, Romania: Matrix Rom, Oct. 2019, pp. 30–37.

[6] *Oculus VR. Medium*. Accessed: May 10, 2021. [Online]. Available: https://www.oculus.com/medium/

[7] *TenkLabs. Kodon*. Accessed: May 10, 2021. [Online]. Available: https://www.tenklabs.com/

[8] C.-W. Chen, J.-W. Peng, C.-M. Kuo, M.-C. Hu, and Y.-C. Tseng, "Ontlus: 3D content collaborative creation via virtual reality," in *MultiMedia Modeling*. Cham, Switzerland: Springer, 2018, pp. 386–389.

[9] *Mojang. Minecraft*. Accessed: May 10, 2021. [Online]. Available: https://www.minecraft.net/

[10] *System Era Softworks*. Astroneer. Accessed: May 10, 2021. [Online]. Available: https://astroneer.space/

[11] *Unity Technologies. Unity 3D*. Accessed: May 10, 2021. [Online]. Available: https://unity.com/

[12] *Epic Games. Unreal Engine 4*. Accessed: May 10, 2021. [Online]. Available: https://www.unrealengine.com/

[13] C. C. L. Wang and Y. Chen, "Layered depth-normal images: A sparse implicit representation of solid models," 2010, *arXiv:1009.0794*. [Online]. Available: http://arxiv.org/abs/1009.0794

[14] *Solidworks Corp. Solidworks*. Accessed: May 10, 2021. [Online]. Available: https://www.solidworks.com/

[15] Autodesk Inc. *Inventor*. Accessed: May 10, 2021. [Online]. Available: https://www.autodesk.com/products/inventor/overview

[16] K. J. Versprille, "Computer-aided design applications of the rational B-spline approximation form," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Syracuse Univ., Syracuse, NY, USA, 1975, Art. no. AAI7607690.

[17] *Robert McNeel & Associates*. Rhino. Accessed: May 10, 2021. [Online]. Available: https://www.rhino3d.com/

[18] L. Stănculescu, R. Chaine, and M.-P. Cani, "Freestyle: Sculpting meshes with self-adaptive topology," *Comput. Graph.*, vol. 35, no. 3, pp. 614–622, Jun. 2011.

[19] Autodesk Inc. *Maya*. Accessed: May 10, 2021. [Online]. Available: https://www.autodesk.com.tw/products/maya/overview

[20] MAXON Computer. *Cinema 4D*. Accessed: May 10, 2021. [Online]. Available: https://www.maxon.net/cinema-4d

[21] Pixologic. *Zbrush*. Accessed: May 10, 2021. [Online]. Available: https://pixologic.com/

[22] Pilgway. *3D-Coat*. Accessed: May 10, 2021. [Online]. Available: https://3dcoat.com/

[23] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, 1987.

[24] M. Becher, M. Krone, G. Reina, and T. Ertl, "Feature-based volumetric terrain generation and decoration," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 2, pp. 1283–1296, Feb. 2019.

[25] É. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez, "Interactive example-based terrain authoring with conditional generative adversarial networks," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 1–13, Nov. 2017.

[26] G. Cordonnier, M.-P. Cani, B. Benes, J. Braun, and E. Galin, "Sculpting mountains: Interactive terrain modeling based on subsurface geology," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 5, pp. 1756–1769, May 2018.

[27] A. T. Galyean and F. J. Hughes, "Sculpting: An interactive volumetric modeling technique," in *Proc. SIGGRAPH*, New York, NY, USA: Association for Computing Machinery, 1991, pp. 267–274.

[28] E. Ferley, M.-P. Cani, and J.-D. Gascuel, "Practical volumetric sculpting," *Vis. Comput.*, vol. 16, no. 8, pp. 469–480, Dec. 2000.

[29] K.-L. Perng, W.-T. Wang, M. Flanagan, and M. Ouhyoung, "A real-time 3D virtual sculpting tool based on modified marching cubes," in *Proc. Int. Conf. Artif. Reality Teleexistence*, 2001, pp. 64–72.

[30] E. Ferley, M.-P. Cani, and J.-D. Gascuel, "Resolution adaptive volume sculpting," *Graph. Models*, vol. 63, no. 6, pp. 459–478, Nov. 2001.

[31] J. A. Bærentzen, "Octree-based volume sculpting," in *Proc. IEEE Vis.*, New York, NY, USA: ACM Press, Oct. 1998, pp. 9–12.

[32] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth, "Hierarchical RLE level set: A compact and versatile deformable surface representation," *ACM Trans. Graph.*, vol. 25, no. 1, pp. 151–175, Jan. 2006.

[33] M. B. Nielsen and K. Museth, "Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets," *J. Sci. Comput.*, vol. 26, no. 3, pp. 261–299, 2006.

[34] S. Lefebvre and H. Hoppe, "Perfect spatial hashing," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 579–588, Jul. 2006.

[35] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–11, Nov. 2013.

[36] K. Museth, "VDB: High-resolution sparse volumes with dynamic topology," *ACM Trans. Graph.*, vol. 32, no. 3, pp. 1–22, Jun. 2013.

[37] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 165–174.

[38] D. Tang, S. Singh, A. P. Chou, C. Hane, M. Dou, S. Fanello, J. Taylor, P. Davidson, G. O. Guleryuz, Y. Zhang, S. Izadi, A. Tagliasacchi, S. Bouaziz, and C. Keskin, "Deep implicit volume compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1293–1303.

[39] E. S. Lengyel, "Voxel-based terrain for real-time virtual simulations," Ph.D. dissertation, Dept. Comput. Sci., Univ. California Davis, Davis, CA, USA, 2010.

[40] Voxel Plugin. *Voxel Plugin—Dynamic Terrain & Procedural Generation in Unreal Engine*. Accessed: May 10, 2021. [Online]. Available: https://voxelplugin.com/
[41] E. Lengyel, "Transition cells for dynamic multiresolution marching cubes," *J. Graph., GPU, Game Tools*, vol. 15, no. 2, pp. 99–122, May 2010.
[42] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of Hermite data," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 339–346, Jul. 2002.
[43] R. Geiss, "Generating complex procedural terrains using the GPU," in *Proc. GPU Gems*, 2008, pp. 7–37.
[44] F. Sarah Frisken Gibson, "Constrained elastic surface nets: Generating smooth surfaces from binary segmented data," in *Proc. 1st Int. Conf. Med. Image Comput. Comput.-Assist. Intervent (MICCAI)*. Berlin, Germany: Springer, 1998, pp. 888–898.
[45] S. Schaefer and J. Warren, "Dual marching cubes: Primal contouring of dual grids," in *Proc. 12th Pacific Conf. Comput. Graph. Appl., (PG)*, Oct. 2004, pp. 70–76.
[46] W. Bolier, W. Hürst, G. van Bommel, J. Bosman, and H. Bosman, "Drawing in a virtual 3D space–introducing VR drawing in elementary school art education," in *Proc. 26th ACM Int. Conf. Multimedia*, Oct. 2018, pp. 337–345.
[47] A-Frame. *A-Painter*. Accessed: May 10, 2021. [Online]. Available: https://aframe.io/a-painter/

CHIEN-WEN CHEN received the B.S. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University (NCKU), Tainan, Taiwan, in 2017. He is currently pursuing the Ph.D. degree with the Multimedia Information System Laboratory, Graduate Program of Multimedia Systems and Intelligent Computing, NCKU and Academia Sinica. His research interests include computer graphics and virtual reality.

MIN-CHUN HU (Member, IEEE) received the B.S. and M.S. degrees in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2004 and 2006, respectively, and the Ph.D. degree from the Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan, in 2011. From 2012 to 2017, she was an Assistant Professor with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, where she was promoted as an Associate Professor, in 2018. She is currently an Associate Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu. She has published more than 90 papers in international journals and conferences. Her research interests include digital signal processing, multimedia content analysis, machine learning, computer vision, computer graphics, virtual reality, and augmented reality. She was awarded the Exploration Research Award from Pan Wen Yuan Foundation, the Outstanding Youth Award from the Computer Society of the Republic of China (CSROC), and the Best Young Professional Member Award of IEEE Tainan Section, in 2015, 2017, and 2018.

WEI-TA CHU (Senior Member, IEEE) received the B.S. and M.S. degrees in computer science from National Chi Nan University, Taiwan, in 2000 and 2002, respectively, and the Ph.D. degree in computer science from National Taiwan University, Taiwan, in 2006. He was a Professor with National Chung Cheng University, from 2007 to 2019. He was a Visiting Professor at Nagoya University, in 2017, and Columbia University, in 2008. He is currently a Professor with the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan. His research interests include multimedia analysis, computer vision, and deep learning. He serves as a TPC Member for IEEE ICME 2022, ACM ICMR 2021, MMM 2020, and IEEE MMSP 2019. He won the Best Full Technical Paper Award in ACM Multimedia, in 2006. He was awarded the Outstanding Youth Electrical Engineer Award by the Chinese Institute of Electrical Engineering, in 2017, the Distinguished Alumni Award presented by National Chi Nan University, in 2014, the Best GOLD Member Award presented by IEEE Tainan Section, in 2013, K. T. Li Young Researcher Award presented by the Institute of Information and Computing Machinery, in 2012, and the Young Faculty Awards presented by National Chung Cheng University, in 2011. He serves as the Publication Co-Chair for ACCV 2020, IEEE VCIP 2018, and ICS 2016. He was an Associate Editor of *IEICE Transactions on Information and Systems*, from 2016 to 2020.

JUN-CHENG CHEN (Member, IEEE) received the B.S. and M.S. degrees in computer science and information engineering from National Taiwan University, Taiwan, in 2004 and 2006, respectively, advised by Prof. Ja-Ling Wu, and the Ph.D. degree in computer science from the University of Maryland, College Park, USA, in 2016, advised by Prof. Rama Chellappa. From 2017 to 2019, he was a Postdoctoral Research Fellow at the Institute for Advanced Computer Studies, University of Maryland. He is currently an Assistant Research Fellow at the Research Center for Information Technology Innovation, Academia Sinica. His research interests include computer vision, machine learning, deep learning, and their applications to biometrics, such as face recognition/facial analytics, activity recognition/detection in the visual surveillance domain. He was a recipient of the ACM Multimedia Best Technical Full Paper Award, in 2006.