Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 2274 – 2289

doi:10.3906/elk-1401-259

A real-time virtual sculpting application by using an optimized hash-based octree

Gülüzar ÇİT*, Kayhan AYAR, Cemil ÖZ
Department of Computer Engineering, Faculty of Computer and Information Sciences, Sakarya University, Sakarya, Turkey

Received: 28.01.2014 • **Accepted/Published Online:** 08.07.2014 • **Final Version:** 15.04.2016

Abstract: The aim of this study is to develop a new 3D virtual sculpting system by using a hash-based octree data structure with an optimized approach to reduce the huge memory and computation cost and also the tree traversal time. The system first reads the selected 3D virtual object model, which must be in a triangular watertight polygon mesh format. Then both the surface and the interior volume of the virtual object is voxelized in order to generate its volumetric dataset. Afterwards, a hash-based data structure with a novel optimized approach is applied to the dataset. After each sculpting process applied to the virtual object, its surface is locally reconstructed and a feedback force is calculated synchronously. Finally, to give the user a realistic sense of interaction, the feedback force is sent back via a haptic device.

Creation durations of a traditional octree, a hash-based octree, and our optimized hash-based octree are compared. Tree traversal and surface reconstruction times of our optimized hash-based octree approach are also compared with the pointer-based one in off-line and real-time while sculpting a 3D model. It is shown that the memory requirement and durations of our optimized approach are the lowest.

Key words: Virtual sculpting, voxelization, octree, hashing, haptics

1. Introduction

Engineers and designers have been using digital computers for years for three-dimensional (3D) modeling. In the '90s the speed and capacity of computers were increased and they also became cheaper. This technological development allowed artists to design and create their works using 3D modeling software. However, today's 3D modeling software does not have the perception of reality and interaction skills like virtual reality environments have. To overcome these limitations, virtual environments have been developed to allow 3D modeling by using virtual reality tools. Generally, this field of virtual reality is called virtual sculpting.

Zhang defines virtual sculpting as the process of interactively creating 3D models by carving a workpiece on the user's computer screen like a real sculptor can do on a piece of clay, wax, or wood [1].

The main purpose of a virtual sculpting system is to allow designers to create and reshape 3D free-form objects via an interactive environment equipped with virtual reality hardware and software. A virtual sculpting system gives realistic feedback of the virtual environment to the user by virtual reality hardware such as haptic devices (e.g., data gloves), head mounted displays, and shutter glasses. Furthermore, the interaction between these devices and the user is performed by the application software.

Virtual sculpting applications are available in many areas especially for education purposes including design and modelling [2–4], virtual medicine [5], and dental surgery [6].

*Correspondence: gulizar@sakarya.edu.tr

In this paper, we developed a new real-time virtual sculpting system by using a hash-based octree data structure with an optimized approach. In this way, we aimed to reduce the huge memory and computation cost and also tree traversal time. Our 3D model is a watertight triangular mesh model and initially both the surface and interior volume of the input model is voxelized in order to generate a volumetric dataset. Then an optimized hash-based octree data structure is constructed in order to reduce the memory requirement needed to store the 3D volumetric dataset and decrease the tree traversal time. After removing material from the virtual workpiece, its surface is locally reconstructed by using the marching cubes algorithm [7], known as the most popular isosurface extraction algorithm. The user interacts with the model via a haptic device to get the feedback force like real-life sculpting.

Our main contribution in octree-based virtual sculpting systems is using an optimized hash-based octree data structure approach in order to store the 3D volumetric dataset instead of using a pointer-based one. By using this optimized data structure, we decreased the memory cost and computation time to build the octree in comparison with other octree representations and also the traversal time needed in collision detection and so the local surface reconstruction time.

The remainder of the paper is organized as follows: a literature review of voxel-based virtual sculpting systems is given in Section 2. We describe the system architecture, hardware, and software parts in Section 3. Then we present the system components of our virtual sculpting system as graphics rendering, simulation, haptic rendering modules in Section 4. Our novel optimized hash-based octree data structure is also explained in detail in Section 4. Experimental results and comparisons of our novel hash-based octree data structure are given in Section 5. Last, we conclude and reveal some future works in Section 6.

2. Literature review

Virtual sculpting applications use many different approaches in the literature, and to construct a virtual sculpting system there are many questions to be answered, such as its input model type, how the data is stored in memory, how the surface is reconstructed, whether it is haptic-based or not, if it is haptic-based, the device type, and how the feedback force is calculated. As we focus on memory optimization and real-time interaction in our haptic-based virtual sculpting application, we propose an optimized hash-based octree volumetric approach aided by a haptic device. Therefore, relevant literature will be discussed below.

The first application of a volumetric sculpture in the literature was proposed by Galyean and Hughes in 1991 [2]. In their study, the initial model defined in a uniform discrete voxel grid is edited interactively to create 3D freeform shapes. Basic operations like addition or subtraction, and several tool definitions such as heat gun, sand paper, or color modifier are proposed. Kaufman and Wang created another sculpting system where the tools are based on carving and sawing [8].

In these studies mentioned above, uniform voxel representation is used where voxels are represented in a 3D discrete array, by dividing 3D space into equal-sized unit voxels. These voxels are labelled according to whether they are inside the object or not. In this representation, when resolution is increased, memory cost also increases because of the growth in the number of voxels needed to display the virtual object. For example, in a $1024 \times 1024 \times 1024$ resolution, a billion voxels are needed to represent a virtual object. Recently, to cope with this huge amount of volume data in 3D virtual models, researchers started to use hierarchical data structures.

In order to work at higher resolutions with lower memory cost, Barentzen preferred to use an octree data structure, which recursively splits voxel-based volumetric data into eight equal octants [9]. Operations are addition and subtraction and the tool shape is a sphere. Ferley et al. also worked on a hierarchical organization

of voxels: a cell can be divided into 27 ones [10,11]. It permits a high level of detail and the tool is defined by an ellipsoid.

Raffin et al. proposed a combination of a volumetric coding and B-rep in their virtual sculpting application [12]. Here, volumetric coding is well adapted to express sculpture operation and, on the other hand, a B-rep extracted from the volumetric coding enables a fast display of the shape and the interactive modification of the viewpoint. A multiresolution approach based on an octree to decrease the memory cost is proposed and both sculpture and tools have been coded as an octree. Interactions are controlled via a spaceball, a keyboard, and a mouse.

Heurtebise and Thon presented a multiresolution model based on 3D Haar wavelets to represent both the sculpture object and the tool as a discrete set of uniform voxels by using the hierarchical structure of Haar wavelet transformations and for each level of detail by applying them in x, y, and z directions recursively [13]. Thus, they took the advantage of level of details to speed up the display, interaction, and sculpture operation times, as uniform spatial enumeration is expensive on processing and display times.

Then, to eliminate the disadvantage of using 3D discrete arrays on memory cost and computation time issues, Heurtebise and Thon proposed a new multiresolution model that combines octree and wavelet; a 3D object is roughly sampled in an octree, where each leaf containing data is thinly sampled thanks to a 3D Haar wavelet transform [14].

Afterwards, they proposed a min/max octree called a "repartition structure" to deal with very large and/or detailed objects and to enhance the collision detection between the tool and the sculpture object [15].

Inspired by all these studies mentioned above, we proposed a real-time virtual sculpting system by using a hash-based octree data structure [16]. Here, we propose an extension of this model that uses a novel optimized hashed-octree approach to decrease the huge memory and computation cost and also to speed up the tree traversal time. In this way, we took advantage of both the hierarchical structure of the octree and also we provided a more rapid tree traversal with the help of hash tables. Furthermore, with the contribution of our optimized approach, we further reduced the memory and computation costs compared with the traditional ones.

3. System architecture

A block diagram of our haptic-based virtual sculpting system architecture is shown in Figure 1. The whole system is composed of two parts: software and hardware. The hardware part includes a computer and a haptic device. The software part consists of three modules: a haptic rendering module, a graphics rendering module, and a simulation module. The details of our system components are described below in detail.

3.1. Hardware

The computer included in our system performs the communication between the haptic device and the software. The haptic device is provided by Geomagic Phantom Omni. This 6-dof haptic device is chosen due to the fact that it is economical and easy to implement.

The stylus of the haptic device directs the virtual tool and when the stylus of the virtual tool collides with the surface of the virtual object, a feedback signal is calculated and this interactive force is sent back to the user through the haptic device.

3.2. Software

As mentioned above, the software part consists of three modules: a graphics rendering module, a haptic rendering module, and a simulation module.

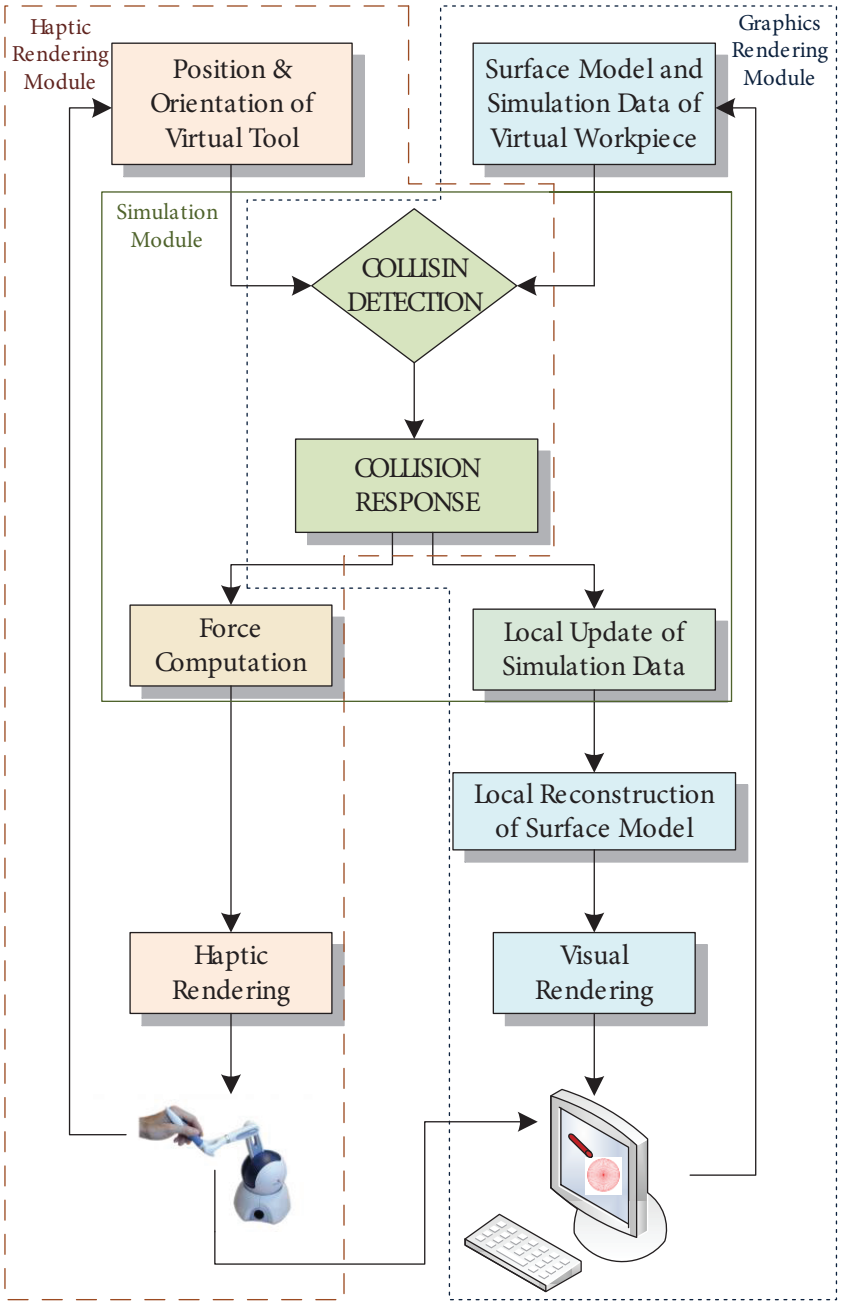

Figure 1. Architecture of our proposed haptic-based virtual sculpting system.

The graphics rendering module is based on rendering of the 3D virtual scene named workspace, sculpting tool, and the sculpture object named workpiece. The physical attributes of the workspace, sculpting tool, and workpiece, such as color, friction, and stiffness, are also described in the initial phase of this module.

The haptic rendering module calculates the feedback force to be sent back to the haptic device. When the virtual probe attached to the stylus of the haptic device collides with the surface of the workpiece, the position and orientation of the probe are provided, the feedback force is calculated, and it is sent back to the user via the haptic device.

Both the graphics and haptic rendering modules are triggered by the simulation module. As soon as the stylus of the haptic device collides with the surface of the sculpture object, the loop of this module starts. This effect is called "collision" and when the collision is detected, a response to this collision occurs in both the haptic and the graphics rendering modules. First, the simulation module calculates the local voxel area affected by the collision between the tool and the workpiece and these affected voxels are deleted from the optimized hash-based octree data. Then the isosurface is locally reconstructed on this local area by using the marching cubes algorithm [7]. Second, the simulation module calculates and sends a feedback signal to the user in the form of an interactive force through the Phantom Device. Depending on the physical attributes of the virtual workpiece and the virtual tool, this force is calculated by the Open Haptics Toolkit.

The initial step of our haptic-based virtual sculpting system software is to convert the input triangular mesh model data into voxel-based volumetric data. Because voxel data is axis-aligned, view-independent, easy and fast to achieve Boolean operations like addition and subtraction, the simulation module works on volumetric data while the graphics rendering module works on triangular meshes. Afterwards, a hash-based octree data structure with a novel optimized approach is constructed from the huge volumetric dataset to reduce the memory cost and the tree traversal time.

The software is implemented by using C++; the scene and all its members are provided by using OpenGL.

4. System components

4.1. Graphics rendering module

This module contains two main stages: preprocessing stage and the graphics rendering loop stage. The preprocessing stage voxelizes the input triangular mesh model, creates the dual cells from the 3D voxel grid, and finally constructs the optimized hash-based octree representation. The graphics rendering loop stage renders the graphics depending on the optimized hash-based octree volumetric data in a frequency of 30 Hz.

4.1.1. Preprocessing

The preprocessing stage includes voxelization as the conversion of the input triangular mesh model into 3D volume data, voxel-cell duality as the creation of the dual cells of each voxel, and the optimized hash-based octree construction as the creation of the hierarchical 3D volume data structure. These three stages are described in detail.

4.1.1.1. Voxelization

Voxelization is concerned with converting geometric objects from their continuous geometric representation into a set of voxels that best approximates the continuous object. As this process mimics the scan-conversion process that pixelizes/rasterizes 2D geometric objects [17], it is also referred to as 3D scan-conversion [18]. Briefly, this conversion process can be defined as representing the geometric object as the set of voxels in a 3D discrete mesh.

The basic idea of voxelization algorithms is to determine whether each voxel is covered by the geometric object or not. Accordingly, if the voxel is covered by the object, the value of the relevant voxel is assigned to "1"; otherwise, its value is assigned to "0". This kind of voxelization approach using a Boolean value to represent only the existence of the object is referred to as binary voxelization. Figure 2 shows resulting binary voxel representations of the Stanford bunny model on three different resolutions. Figures 2a–2c show the model on $64 \times 64 \times 64$, $128 \times 128 \times 128$, and $256 \times 256 \times 256$ resolutions, respectively.

Polygonal models have only the surface information of the geometric model. However, we can access both the surface and interior volume information of the 3D model by using voxel-based data. Furthermore,

Boolean operations like addition and subtraction are also easier and faster by using voxel-based data rather than polygonal data. For all these reasons, there are many existing voxelization algorithms in the literature that convert geometric models into their corresponding voxel-based models, such as surface-based, volume-based, binary, and multivalued [19]. A model can be a combination of these models as well (e.g., surface-based binary volume model). In our virtual sculpting application, we need the whole volume information of the input geometric model but our corresponding voxel-based volumetric data structure will not be used for rendering. Therefore, to achieve a volume-based binary voxel-data structure, we voxelize our input triangular mesh model by using Möller's 3D triangle-box overlap test [20,21] to voxelize its surface and by using flood filling [22] to fill its interior volume as seen in the following algorithm steps.

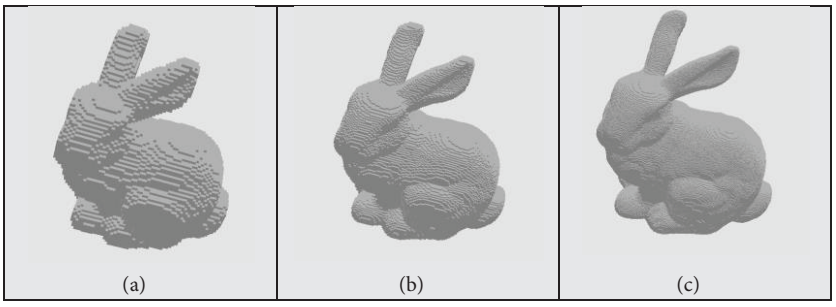

Figure 2. Binary voxel representations of the Stanford bunny model on (a) 64 × 64 × 64, (b) 128 × 128 × 128, and (c) 256 × 256 × 256 resolutions.

Algorithm 1. Get 3D volumetric binary voxel-dataset from input triangular mesh model

```
Calculate the AABB of the 3D input triangular mesh model
Create a discrete voxel grid in this bounding box in the desired resolution and flag each voxel as '0'
// Surface Voxelization
FOR all triangles in the model DO
        Calculate AABB
        FOR all voxels in this AABB DO
                IF relevant voxel's flag is '0' THEN
                        Apply Möller's 3D triangle-box overlap test.
                        IF triangle and voxel overlap THEN
                                Convert this relevant voxel's flag as '1'
// Interior Voxelization
Fill the interior volume of the 3D voxel data by using 3D flood-filling algorithm
```

Möller's 3D triangle-box overlap test algorithm checks if a triangle fully overlaps a box or not. This algorithm is based on separating axis theorem (SAT) and it has 13 criteria to test if there is a separating axis between the triangle and the box. If all tests pass, i.e. if there is no separating axis, the triangle overlaps the box.

The majority of traditional voxelization algorithms use only the center of the voxels for polygon overlap testing. In contrast, here the whole volume of a voxel is tested. Thus, the accuracy of the surface voxelization is higher than that of the other voxelization algorithms. On the other hand, elapsed time during voxelization is longer than that of the others because of the number of test criteria. However, this overlap test algorithm is suitable for parallelism because of testing each triangle individually. Thus, this disadvantage in the voxelization process can be eliminated by using multithreading [23].

The second stage of the preprocessing is the construction of voxel-cell duality. After voxelizing the triangular mesh model, depending on the adjacent voxel values of the corresponding voxel, each voxel area of the resulting 3D binary volume data is converted into its dual cell structure.

4.1.1.2. Voxel-cell duality

A voxel is a cube in the 3D space and its value is the same throughout its whole volume. A cell in the 3D space is a cube as well, but it is a combination of eight adjacent voxels. These voxel values are assigned to each of the cell's corresponding vertex as seen in Figure 3a. We create this cell structure by shifting each voxel's value to its upper-back-right vertex position by an approach similar to that in Knoll et al. [24] as seen in Figure 3b. Therefore, each vertex of the cell is assigned a binary value depending on the corresponding voxel and finally the cell value is assigned an unsigned char value by combining these binary values like $V_0V_1V_2V_3V_4V_5V_6V_7$. For example, if the eight vertex value of a cell is 10110010, correspondingly its decimal value is assigned to 178, because the decimal equivalent of 10110010 in binary base is 178. In this way, before the construction of the optimized hash-based octree, we remap our 3D volume grid and thereafter cells can be used instead of voxels.

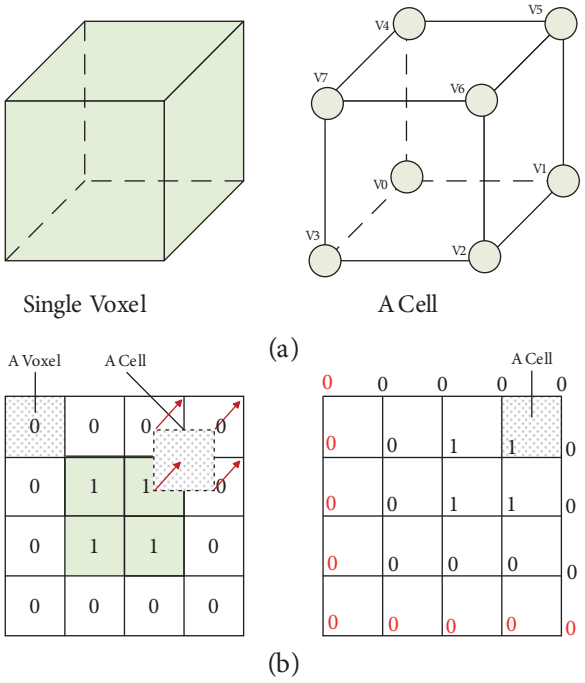

Figure 3. (a) A single voxel and a cell, (b) 2D representation of voxel-cell duality: 2D grid (left) and its dual cell grid (right)

This mapping is needed for surface reconstruction and can be done on either in the preprocessing stage as we have done or during real-time interaction. Mapping voxel values into cells during real-time interaction is not suitable for time-critical applications. Therefore, by doing this mapping in preprocessing time instead of real-time, we aim to accelerate the duration of global and local surface reconstruction and so the graphics rendering module.

The last stage of the preprocessing is the construction of the optimized hash-based octree data structure. After voxelizing the triangular mesh model and creating the cell structure from voxels, the resulting 3D

binary volume data are converted into an optimized hash-based octree data structure to decrease the memory requirement, computation, and tree traversal time.

4.1.1.3. Optimized hash-based octree construction

3D discrete voxel representation is the easiest and first method of representing voxels where each voxel is an equal-sized cube in 3D space. However, when the resolution increases, the number of voxels needed to represent the object also increases as does the memory cost. For instance, in 1024 $\times$ 1024 $\times$ 1024 resolution, more than one billion voxels are needed to represent the 3D object. Memory cost is a critical problem in voxel-based applications and to optimize this cost new hierarchical data structures dividing the space recursively into subregions are proposed, such as BSP tree, quadtree, and octree [25]. In virtual sculpting applications, the most preferred one among them is octree [9,10,13]. Because of this, we also decided to use octree.

The octree starts with a root node representing the whole space of the tree. Each node of the octree represents a subregion of space and has eight child octants divided equally by three directions in 3D space.

Several methods are proposed for storing octree data in memory. Linear and pointer-based methods are the most common [26]. A linear octree only stores leaf nodes, i.e. the location codes of the voxels, consecutively in memory. Therefore, pointers are not needed in linear octree methods. In this way, we can save memory and store each node by encoding them. However, traversing the tree to reach a node in a linear octree is a complex and time consuming process. Instead, in pointer-based octree methods, every parent node keeps their eight child nodes' addresses. These child nodes, named octants, can represent space as empty, full, or partially full depending on how the model covers it. In this type of octree data structure, traversing and reaching a node in the tree is easier and faster than in a linear octree. Furthermore, memory is saved, because there is no need to store each voxel in 3D space as in the 3D discrete grid. These are basic reasons to prefer pointer-based octree in sculpting applications.

In the present study, we proposed to use an optimized hash-based octree data structure instead of the traditional pointer-based octree for storing voxel-based volumetric data. The goal was focusing on decreasing the huge memory cost and computation cost needed for storing 3D volumetric voxel data and also the interaction time to make the sculpting process in real-time.

A hash-based octree is another way of representing an octree by using a hash table [27]. Thus, it allows access to nodes directly instead of traversing in pointer-based octree complex hierarchy. In this octree representation, each node is assigned a key value that defines the node itself and its location code of the three axes (x,y,z) [28]. Keys are used to access the nodes in memory instead of pointers. These values are calculated from the location code of the node. Each octant of the octree has a 3-bit label and in each level of the tree the order of this label is the same. The order is 000, 001, 010, 011, 100, 101, 110, and 111, respectively, as seen in Figure 4.

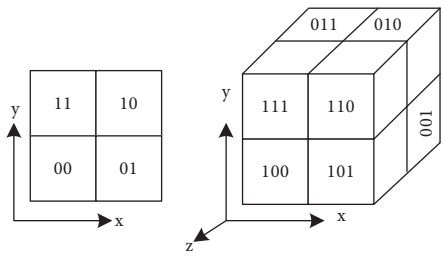

Figure 4. Labels of each node in a given depth in a quadtree in 2D (left) and an octree in 3D space (right).

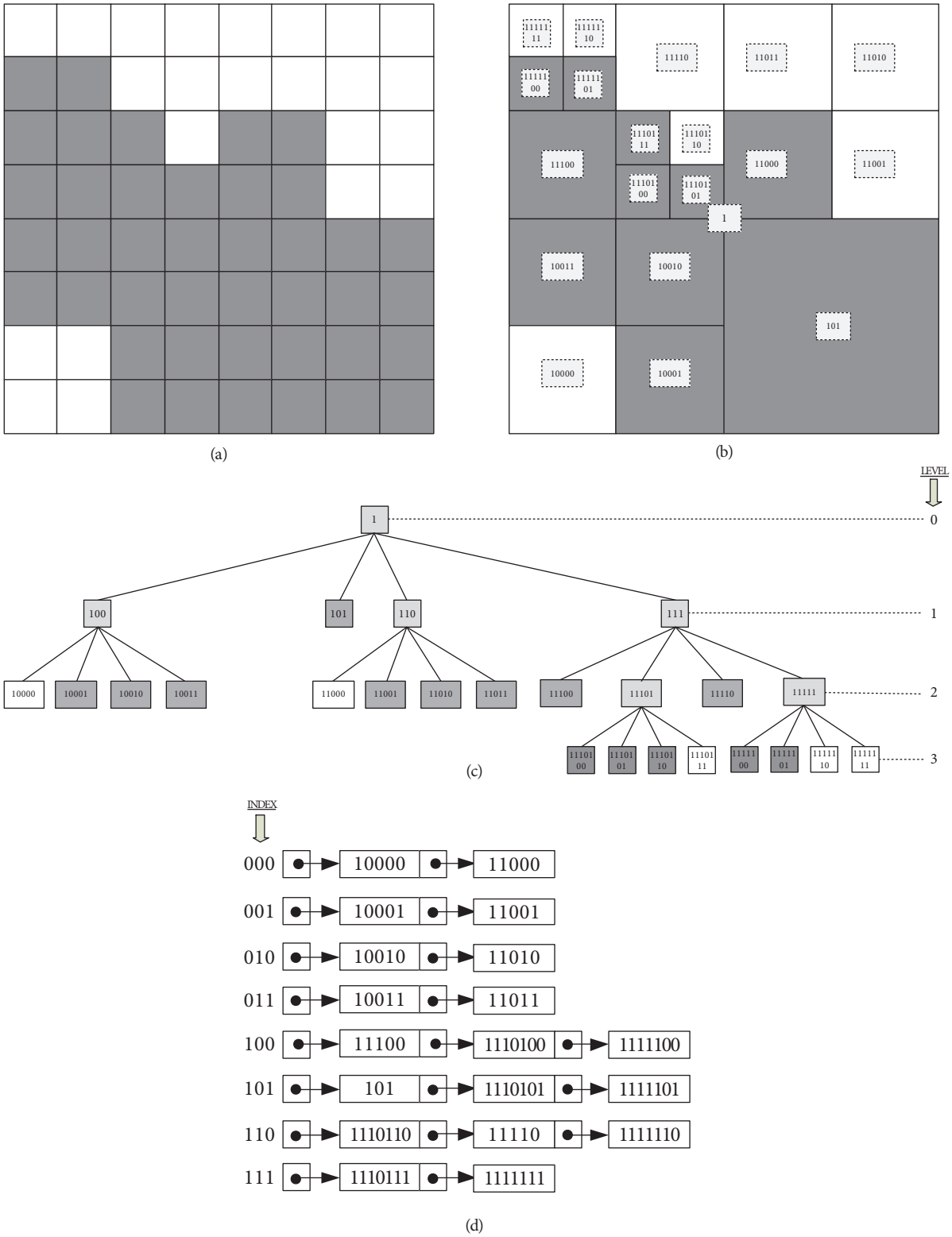

Figure 5. (a) 2D 8 × 8 binary shape example, (b) its quadtree representation and corresponding keys of nodes (traversal direction is bottom-left, bottom-right, top-left, top-right), (c) pointer-based octree representation, and (d) our optimized hash-based octree representation.

Figure 5a shows an example of a 2D binary shape with a resolution of 8 × 8, Figure 5b shows its quadtree representation and corresponding keys of nodes, Figure 5c shows its pointer-based octree representation, and finally Figure 5d shows our optimized hash-based octree representation.

As seen in Figure 5d, we can optimize the hash table by adding only the leaf nodes, i.e. both completely full nodes and empty nodes, to the hash table. There is no need to add parent nodes to the table having child nodes. If needed, we can access information of these nodes from their child nodes. By doing this, we decrease the total node count, and so the memory requirement as well.

The dimension of the hash table is defined depending on the resolution of the octree as powers of two (2^n, $n = \{0,1,2,\dots\}$) and each node is assigned to an index in the hash table by looking at the right 2^n bits from its key in a bottom-up octree manner.

Initially, the level of the root node is assigned to 0 and increased one by one until a predefined resolution (see Figure 5c). The key of the root node is assigned to 1 (see Figure 5b). The key of a node in level *1* is calculated by adding 3 bits to the right of the parent node's key depending on its location code in the octree. This continues by adding 3 bits to its right in each level. More generally, the key $k(n)$ of node n is calculated by dilating and interleaving the location code of the associated node and adding 1 before its leftmost bit. Interleaving is the process of dilating two or more integers, shifting them so the significant bits of each integer are aligned with the inserted zeros in the other integers and then bitwise OR-ing them together [29], while dilating is the process of inserting a number of zeros before each of its bits [30]. Thus, the generated key for a given level of l and location code of x, y, z is as follows:

$$DIL\,(x) = x_l x_{l-1} x_1 x_0 \tag{1}$$

$$DIL\,(y) = y_l y_{l-1} y_1 y_0 \tag{2}$$

$$DIL\,(z) = z_l z_{l-1} z_1 z_0, \tag{3}$$

where $x = x_l x_{l-1} x_1 x_0$, $y = y_l y_{l-1} y_1 y_0$, and $z = z_l z_{l-1} z_1 z_0$ in bit format and *DIL(x)*, *DIL(y)*, and *DIL(z)* are nonuniform dilations of x, y, and z with 1 bit of location code and 2 zeros, recursively.

$$INT\,(x, y, z) = DIL\,(z) \ll 4 \,|\, DIL\,(y) \ll 2 \,|\, DIL\,(x) \tag{4}$$

$$INT\,(x, y, z) = z_l y_l x_l z_{l-1} y_{l-1} x_{l-1} \dots z_1 y_1 x_1 z_0 y_0 x_0, \tag{5}$$

where *INT(x,y,z)* is the interleaving of *DIL(x)*, *DIL(y)*, and *DIL(z)* in a *zyx* order.

$$KEY\,(x, y, z) = 1 \,\&\, INT(xyz) \tag{6}$$

$$KEY\,(x, y, z) = 1 z_l y_l x_l z_{l-1} y_{l-1} x_{l-1} \dots z_1 y_1 x_1 z_0 y_0 x_0, \tag{7}$$

where *KEY(x,y,z)* is the key and it is generated by adding 1 bit to *INT(x,y,z)* leftmost. In addition, the location code of x, y, z can also be obtained from the key of the related node by executing the reverse processes from 7 to 1.

4.1.2. Graphics rendering loop

The graphics rendering loop is responsible for rendering the 3D scene, which is composed of a workpiece and a sphere-shaped virtual tool at 30 Hz frequency. The surface of the workpiece is reconstructed in the loop by using the marching cubes algorithm depending on the node values in the optimized hash-based octree data structure.

When the virtual probe attached to the stylus of the haptic device collides with the surface of the workpiece, the local area in the optimized hash-based octree data affected by this collision is recalculated. Then the marching cubes algorithm [7] is reapplied and the surface of the workpiece is reconstructed again.

Figure 6 shows two extracted isosurfaces from 2D cell grids and their optimized quadtree representations before (Figure 6a) and after (Figure 6b) interactions between the object and the tool. As seen in the figure, after collision, the cell colliding with the tool is assigned to 0 and this also affects 5 adjacent cells to this colliding cell, i.e. left, right, top, top-left and top-right cells. Afterwards, the optimized hash table is also updated depending on the new cell values.

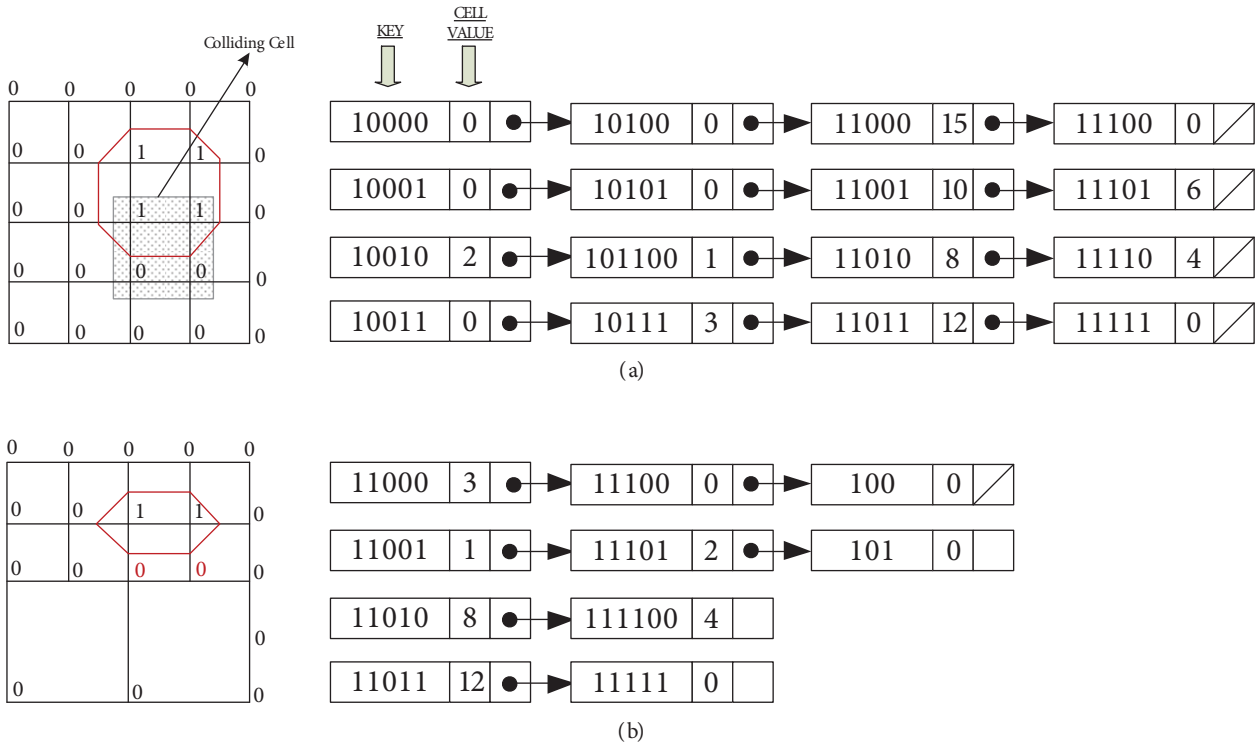

Figure 6. Extracted isosurface from 2D cell grid and corresponding optimized hash-based quadtree representation before (a) and after (b) collision.

4.2. Simulation module

The simulation module has two stages: collision detection and collision response. These stages are described in detail below.

4.2.1. Collision detection

Collision detection is performed by testing a possible collision between the sculpture object's 3D voxel-based volumetric data in the form of an optimized hash-based octree structure and the virtual probe of the haptic

device. The virtual probe is controlled by the stylus of the Phantom Omni Haptic Device and this stage contiguously checks if the stylus of the haptic device's AABB (axis-aligned bounding box) collides with the sculpture object's volumetric data and if the collision occurs. After a collision is detected, the collision response is performed.

4.2.2. Collision response

When a collision occurs between the sculpture object and the stylus of the haptic device, the collision response calculates the feedback force and updates the optimized hash-based octree data. Then it triggers haptic rendering loops and graphics rendering loops one after another. The feedback force calculation is also a part of the haptic rendering module (see Figure 1) and is explained in detail in Section 4.3.1. The local update of the optimized hash-based octree is a part of the graphics rendering loop (see Figure 1) as well and is explained in detail in Section 4.1.2.

4.2.3. Haptic rendering module

The haptic rendering module performs the calculation of the feedback force and it transmits this interactive force to the user through the haptic device.

The stylus of the haptic device directs the virtual tool and when the stylus of the virtual tool collides with the surface of the virtual object, a feedback signal is calculated and this interactive feedback force caused by the signal is sent back to the user through the haptic device.

4.2.4. Feedback force calculation

The physical properties such as stiffness and friction are the same throughout the whole volume of the sculpture object and the feedback force calculated depending on these properties is sent back to the user via a haptic device. Here, we used the easiest haptic rendering module, named point-based haptic rendering [31]. It simulates the stylus of the haptic device as haptic interface point (HIP) to interact with virtual objects and the force F is calculated by spring-damper model as

$$\vec{F} = k\Delta\vec{x}, \tag{8}$$

where k is the stiffness coefficient of the virtual workpiece and Δx is the displacement of the haptic probe from the collided surface point of the workpiece.

5. Experimental results

The results given in this paper were obtained on a PC with an Intel Core 2 Quad 2.5 GHz with 4 GB memory, a ATI Radeon HD 5850 with 1 GB memory, and Windows 7 64 bit.

To determine the performance of our optimized hash-based octree approach on different resolutions, we compare this approach both in preprocessing depending on memory cost, computation, and surface reconstruction times and also in real-time depending on times after collision detection.

Table 1 shows the construction times of pointer-based, hash-based, and our novel approach as an optimized hash-based octree from a 3D discrete voxel grid and number of nodes needed to construct these three data structures, respectively.

Number of nodes needed to construct a hash-based octree is the same as for a pointer-based one as seen in Table 1. Therefore, the memory requirement is the same as well. However, number of nodes needed to construct

our optimized hash-based octree is less than that of the other two approaches, because in our hash-based octree approach only the leaf nodes, i.e. both completely full nodes and empty nodes, are added to the table, not the parent nodes having child nodes. The calculated mean value of this decrease in memory requirement is about

Table 1. Number of nodes in pointer-based, hash-based, and optimized hash-based octree data structures and their corresponding construction times.

Model name	# of triangles	Resolution	# of nodes		Construction time (s)		
			Pointer & hash-based	Optimized hash-based	Pointer-based	Hash-based	Optimized hash-based
Sphere	4800	64 × 64 × 64	50,456	44,150	0.099	0.059	0.034
		128 × 128 × 128	204,560	179,040	0.848	0.414	0.138
		256 × 256 × 256	820,552	718,187	6.724	2.760	0.723
		512 × 512 × 512	3,281,168	2,871,821	38.123	23.320	4.871
Cube	12	64 × 64 × 64	59,848	52,368	0.096	0.070	0.0354
		128 × 128 × 128	250,376	219,080	0.827	0.419	0.146
		256 × 256 × 256	1,024,584	896,512	6.737	3.022	0.815
		512 × 512 × 512	4,145,800	3,627,576	39.895	17.233	5.404
Cylinder	252	64 × 64 × 64	34,736	30,395	0.099	0.051	0.028
		128 × 128 × 128	142,680	124,846	0.810	0.341	0.109
		256 × 256 × 256	579,744	507,494	7.572	2.648	0.626
		512 × 512 × 512	2,334,264	2,042,699	41.232	19.427	4.488
Torus	512	64 × 64 × 64	33,696	29,492	0.104	0.052	0.022
		128 × 128 × 128	134,272	117,510	0.814	0.333	0.107
		256 × 256 × 256	547,360	479,018	6.801	2.636	0.623
		512 × 512 × 512	2,187,464	1,911,792	40.254	19.384	4.745

Table 2. Preprocessing marching cubes durations in the pointer-based and optimized hash-based octree data structures on different resolutions.

Model name	Resolution	Preprocessing MC durations (s)	
		Pointer-based	Optimized hash-based
Sphere	64 × 64 × 64	0.173	0.169
	128 × 128 × 128	1.051	0.593
	256 × 256 × 256	7.146	2.355
	512 × 512 × 512	24.604	9.501
Cube	64 × 64 × 64	0.189	0.173
	128 × 128 × 128	1.154	0.705
	256 × 256 × 256	7.886	2.763
	512 × 512 × 512	23.105	11.475
Cylinder	64 × 64 × 64	0.155	0.103
	128 × 128 × 128	0.891	0.406
	256 × 256 × 256	7.208	1.652
	512 × 512 × 512	14.413	6.601
Torus	64 × 64 × 64	0.132	0.122
	128 × 128 × 128	0.824	0.393
	256 × 256 × 256	5.835	1.567
	512 × 512 × 512	13.714	7.139

14%–15% in different resolutions. Furthermore, as seen in Table 1, the duration of our optimized hash-based octree construction is the least.

Table 2 shows the comparison of our optimized hash-based octree data structure with the pointer-based one according to the preprocessing marching cubes durations on different resolutions and on 4 different well-known 3D watertight models.

The graphics of Figure 7 show the total real-time tree traversal and marching cubes durations in the pointer-based and optimized hash-based octree data structures when making the same hole in the watertight Stanford bunny model in different resolutions. Because of the lower node count and the higher speed of the tree traversal, the optimized hash-based octree data structure has the least processing time among these three data structures in both preprocessing time and real-time as seen in the table and graphics below.

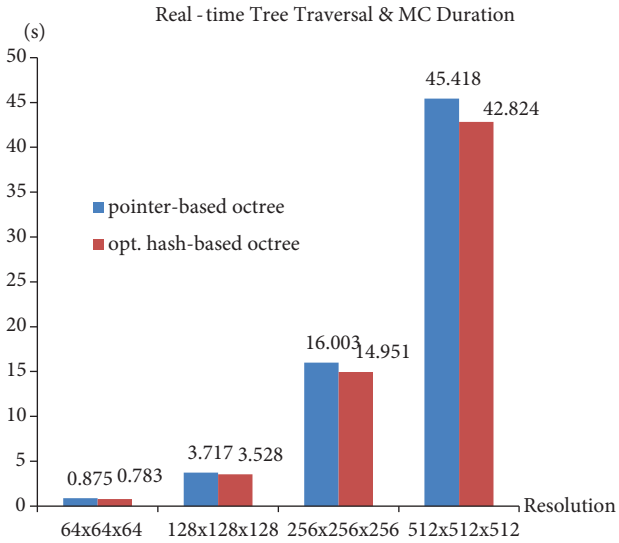

Figure 7. Real-time tree traversal and marching cubes durations in the pointer-based and optimized hash-based octree data structures on the bunny model for the same carving.

Figure 8 shows a screenshot of our virtual sculpting application program after making a hole in one ear of the watertight Stanford bunny model.

6. Conclusions and future work

The paper introduces a new 3D virtual sculpting system by using a hash-based octree data structure with an optimized approach in order to reduce the huge memory cost, computation cost, and the tree traversal time. The construction times of the traditional octree, hash-based octree, and our novel optimized hash-based octree are calculated on different resolutions. The tree traversal and surface reconstruction times are also calculated. Then the results are compared in off-line and real-time and it is shown that the memory requirement and the duration of our optimized hash-based octree data structure approach are lowest.

We are planning some future works to overcome the shortcomings. Our first future goal is smoothing the marching cubes isosurface. Our second goal is make the haptic rendering volume-based instead of point-based with the addition of different sculpting tools.

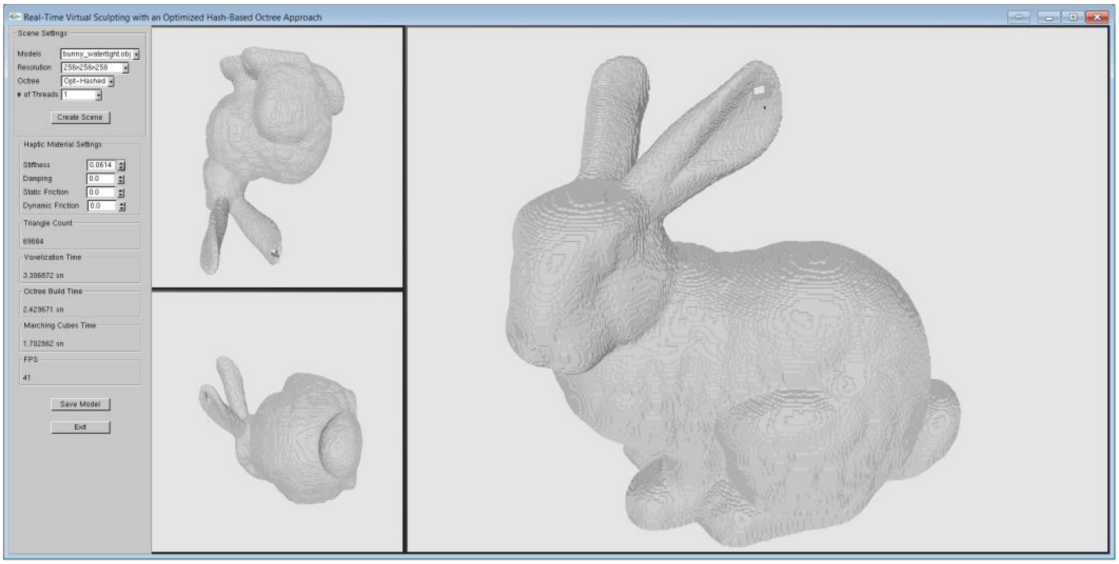

Figure 8. A screenshot of our virtual sculpting application program.

Acknowledgment

This paper was prepared with the support of BAP, Sakarya University, with the grant number 2010-50-02-010.

References

[1] Zhang W. Virtual Prototyping with Surface Reconstruction and Freeform Geometric Modeling Using Level-set Method. PhD, Missouri University of Science and Technology, Rolla, MO, USA, 2008.

[2] Galyean A, Hughes JF, Sculpting: an interactive volumetric modeling technique. Computer Graphics 1991; 25: 268-274.

[3] Perng, KL, Wang WT, Flanagan M, Ouhyoung M. A real-time 3d virtual sculpting tool based on modified marching cubes. In: Proceedings of International Conference on Artificial Reality and Telexistence; 2001. pp. 64-72.

[4] Ho CC, Tu CH, Ouhyoung M. Detail sculpting using cubical marching squares. In: ICAT'05-Proceedings of the 2005 International Conference on Augmented Telexistence; 2005. pp. 10-15.

[5] Niu Q, Chi X, Leu MC, Ochoa J. Image processing, geometric modeling and data management for development of a virtual bone surgery system. Computer Aided Surgery 2008; 13: 30-40.

[6] Yau HT, Tsou LS, Tsai MJ. Octree-based virtual dental training system with a haptic device. Computer-Aided Design & Applications 2006; 3: 415-424.

[7] Lorensen WE, Cline HE, Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics 1987; 21: 163-169.

[8] Wang S, Kaufman AE. Volume sculpting. In: Symposium on Interactive Graphics 1995; ACM SIGGRAPH.

[9] Barentzen A. Octree-based volume sculpting. In: IEEE Visualization '98, Late Breaking Hot Topics Proceedings; 1998; IEEE Computer Society Press. pp. 9-12.

[10] Ferley E, Cani MP, Gascuel JD. Practical volumetric sculpting. The Visual Computer 2000; 16: 469-480.

[11] Ferley E, Cani MP, Gascuel JD. Resolution adaptive volume sculpting. Journal of Graphical Models 2001; 63: 459-478.

[12] Raffin R, Gesquiere G, Remy E, Thon S. VirSculpt: a virtual sculpting environment. In: International Conference Graphicon; 2004. pp. 184-187.

[13] Heurtebise X, Thon S. Discrete tools for virtual sculpture. In: GRAPP'06; 2006. pp. 415-422.

[14] Heurtebise X, Thon S. Multiresolution representation and deformation of very large volume datasets based on Haar wavelets. In: 3rd International Conference on Geometric Modeling and Imaging; 2008. pp. 34-40.

[15] Heurtebise X, Thon S. A repartition structure for collision detection and deformation of discrete objects based on 3D wavelets. International Journal of Computer Information Systems and Industrial Management Applications 2011; 3: 568-577.

[16] Çit G, Ayar K, Serttaş S, Öz C. A real-time virtual sculpting application with a haptic device. Turkic World Mathematical Society Journal of Applied and Engineering Mathematics 2013; 3: 105-112.

[17] Foley JD, Van Dam A, Feiner SK, Hughes JF. Computer Graphics: Principles and Practice. 2nd Edition. Boston, MA, USA: Addison-Wesley, 1990. pp. 92-99.

[18] Kaufman A, Cohen D, Yagel R. Volume Graphics. IEEE Computer 1993; 26: 51-64.

[19] Dong Z, Chen W, Bao H, Zhang H, Peng Q. Real-time voxelization for complex polygonal models. In: Proceedings of 12th Pacific Conference on Computer Graphics and Applications; 2004. pp. 43-50.

[20] Möller AT. Fast 3d triangle-box overlap testing. Journal of Graphics Tools 2001; 6: 29-33.

[21] Möller AT, Haines E. Real-Time Rendering, Natick, MA, USA: AK Peters Ltd, 2002.

[22] Feng L, Soon SH. An effective 3D seed fill algorithm. Comput & Graphics 1998; 22: 641-644.

[23] Öz C, Çit G, Ayar K. Multithreaded voxelization method for virtual sculpting. European Conference of Technology and Society, 2013.

[24] Knoll A, Wald I, Parker S, Hansen C. Interactive isosurface ray tracing of large octree volumes. In: Proceedings of the IEEE Symposium on Interactive Ray Tracing; 2006. pp. 115-124.

[25] Samet H. Applications of Spatial Data Structures. Boston, MA, USA: Adison-Wesley, 1989.

[26] Gargantini I. Linear octrees for fast processing of three dimensional objects. Computer Graphics and Image Processing 1982; 4: 365-374.

[27] Castro R, Lewiner T, Lopes H, Tavares G, Bordignon A. Statistical optimization of octree searches. Computer Graphics Forum 2008; 27: 1557-1566.

[28] Morton GM. A Computer Oriented Geodetic Database and a New Technique in File Sequencing. IBM Ltd, 1966.

[29] Stocco L, Schrack G. Integer dilation and contraction for quadtrees and octrees. In: Proc. IEEE Pacific Rim Conf. Communications, Computers and Signal Processing (PACRIM '95); 1995. pp. 426-428.

[30] Stocco L, Schrack G. On spatial orders and location codes. IEEE Transactions on Computers 2009; 58: 424-432.

[31] Basdogan C, Srinivasan MA. Haptic Rendering in Virtual Environments. Handbook of Virtual Environments: Design, Implementation, and Applications. Boca Raton, FL, USA: CRC Press, 2002. pp. 117-134.