# Graphic Processing Units (GPUs)-Based Haptic Simulator for Dental Implant Surgery

**Fei Zheng**
e-mail: zhengfei@nus.edu.sg

**Wen Feng Lu**
e-mail: mpelwf@nus.edu.sg

**Yoke San Wong**
e-mail: mpewys@nus.edu.sg

Department of Mechanical Engineering,
National University of Singapore,
9 Engineering Drive 1,
Block EA,
Singapore 117576, Singapore

**Kelvin Weng Chiong Foong**
Faculty of Dentistry,
National University of Singapore,
21 Lower Kent Ridge Road,
Singapore 119083, Singapore
e-mail: kelvinfoong@nuhs.edu.sg

*This paper presents a haptics-based training simulator for dental implant surgery. Most of the previously developed dental simulators are targeted for exploring and drilling purpose only. The penalty-based contact force models with spherical-shaped dental tools are often adopted for simplicity and computational efficiency. In contrast, our simulator is equipped with a more precise force model adapted from the Voxmap-PointShell (VPS) method to capture the essential features of the drilling procedure, with no limitations on drill shape. In addition, a real-time torque model is proposed to simulate the torque resistance in the implant insertion procedure, based on patient-specific tissue properties and implant geometry. To achieve better anatomical accuracy, our oral model is reconstructed from cone beam computed tomography (CBCT) images with a voxel-based method. To enhance the real-time response, the parallel computing power of GPUs is exploited through extra efforts in data structure design, algorithms parallelization, and graphic memory utilization. Results show that the developed system can produce appropriate force feedback at different tissue layers during pilot drilling and can create proper resistance torque responses during implant insertion.* [DOI: 10.1115/1.4024972]

*Keywords: haptic rendering, surgical training, dental drilling, implant insertion, voxel model, parallel computing, GPU, VPS*

## 1 Introduction

The surgical procedure in dentistry is guided by the tactile sensation that the dentist perceives through his instrument. Although it is possible to practice the procedure on a cadaver which offers equivalent tactile response, their supply is limited and the cost is quite high. Alternatively, a Manikin-based training simulator provides a plastic model of the patient's head and mouth, on which dental procedures can be performed using actual dental instruments on artificial oral tissues. Some of these simulators, such as DentSim [1] and DSEplus [2], have been commercialized and proven to be helpful in dental education. However, they can hardly duplicate the pathological diversity. In addition, considering the frequent replacement of the artificial oral tissues, the cost is also not cheap. On the other hand, with advances in virtual reality technology, there is much potential in the use of a haptics-based simulator, which employs virtual models of the oral anatomy and a haptic device as a training platform. The trainee holds the stylus of the haptic device to manipulate a set of virtual hand-pieces, while the tactile feedback reproduces clinical sensation during practicing. A haptics-based simulator can be much more cost effective. A particular surgical procedure can be virtually practiced many times, without replacing any physical models. The critical performance requirement is to provide physically realistic haptic simulation, in real-time.

This paper aims at the simulation of dental implant surgery for training purposes. Dental implants are a reliable and long-lasting replacement for missing teeth. Dental implant surgery generally includes three procedures. First, a pilot hole is created in the jawbone. Second, the root part of a dental implant is screwed into the jaw through the pilot hole, taking the place of the root of a missing tooth root. Finally, an abutment is installed on the root part and a custom-made crown is placed over it. Among these three procedures, the first two are the most critical to the success of this surgery. The drilling procedure has to be carefully conducted within a limited space, without damaging the underlying mandibular canal, maxillary sinus, and adjacent tooth roots. Dentists are required to practice and get familiar with the tactile sensation when drilling through different oral tissues, so that they can stop when the drill touches critical but invisible oral structures. Meanwhile, as overscrewing may cause the stripping of the implant, dentists need to develop an intuitive sensation to achieve optimal tightness between the implant and the bone tissue.

To reduce the aforementioned risks during the dental implant surgery, a haptics-based training simulator has been built for novice dentists to practice the pilot drilling procedure and the implant insertion procedure. Both procedures may affect several types of oral tissues, including an exterior layer of hard cortical bone, an interior layer of spongy cancellous bone, the tooth enamel, dentin and pulp. It is of vital importance for the simulator to capture the inhomogeneous features of the involved oral anatomy. In literature, surface-based approaches were mostly employed [3–7], which model the tooth/bone tissues with triangular meshes and compute the contact resistance force based on penetration depth. Although these approaches are effective to provide basic haptic feedback, they are not suitable for surgical simulations involving multiple inhomogeneous tissues, due to the difficulties in modeling the internal structures and various tissue properties. Meanwhile, voxel-based approaches are more convenient and intuitive for modeling the physical properties of different inhomogeneous tissues. Attributes like voxel density, tissue type, position and color can be assigned to a voxel, according to its location in a specific anatomical structure. Corresponding force feedback can be simulated using these attributes, along with the position and orientation of virtual instruments. Voxel-based models have been commonly applied in simulations for dental preparation [8–11], craniotomy [12,13] and bone surgery [14,15]. However, the methods used for the voxel model construction were not the same. Some [10–13] built their models from polygon models, using a particular voxelization method. Others [8,9,14,15] built their models directly from original CT images and reconstructed an isosurface for graphic rendering. The latter method is adopted here as patient-specific tissue properties can be extracted directly from CT images, upon which anatomically correct voxel representations of patients' oral structures can be constructed. Image data are preprocessed to alleviate imaging noise, providing comparable visual results to the former one.

Contributed by the Computers and Information Division of ASME for publication in the Journal of Computers and Information Division in Engineering. Manuscript received November 6, 2012; final manuscript received June 26, 2013; published online September 12, 2013. Assoc. Editor: Krishnan Suresh.

Realistic haptic simulation requires intensive computations for large-scale data management, collision detection, physical-based force computation, and real-time update of object models. However, it is quite difficult to complete of all these tasks at a stable haptic rendering rate (1000 Hz) with traditional methods. Consequently, to ease the computation burden, sphere representations of the virtual instrument and penalty-based force models are often used for simplification [10,11,16]. The VPS force model was first introduced by McNeely et al. [17]. Voxmap is a collection of voxels representing the virtual scene, while PointShell is a set of points distributed around the virtual tool, each having an inward-pointing surface normal to facilitate the calculation of penetration depths. Resistant force is computed by adding the contribution of each point based on its penetration depth. Variations of the original VPS model can also be seen, with improved sampling methods and data structures [13,14,18]. Basically, VPS and its variations can be classified as multiple-point penalty-based models. Although more precise, the feeling is still like a touch resistance force rather than a drilling force that comes from the rotational motion of the cutting edges. In response, a novel drilling force model is proposed, based on drill rotation and contact tissue properties. The new model can also accommodate various drill shapes.

Meanwhile, haptics-based torque modeling can hardly be found in literature. To the best of the authors' knowledge, the Osteosynthesis screw insertion simulator [19] is probably the only haptics-based screw insertion simulator for training purposes. The torque output of this simulator was computed based on a torque-rotation relationship derived from existing experimental data, which were collected during an Orthopaedic surgery performed on a sheep tibia. The current system can provide reasonably realistic screw insertion experience on three types of bone structure, from three rotations before and during the stripping phase. However, this approach is not suitable to model the torque feedback on patient-specific oral tissues, as there is no relation between the torque and the tissue properties. In addition, no graphic display is provided. To tackle this issue, a voxel-based torque model is proposed, based on the tissue properties as well as the implant geometry.

Moreover, to accommodate more precise voxel models and more computationally intensive haptic models, stable force/torque rendering at haptic frequency (1000 Hz) becomes a major challenge. To solve this problem, the parallel computing architecture is exploited. Unfortunately, the parallel computation power of CPUs is limited, as more transistors are devoted to flow control and data caching. Meanwhile, GPUs have more transistors for data processing. This architectural feature makes GPUs more specialized for compute-intensive and highly parallel computations. GPU-based architecture usually guarantees significant speedup but require more complex algorithm design, due to the need to adapt algorithms to the graphic domain. Extra efforts are made to design proper data structures, parallelize the algorithms and utilize appropriate graphic memory. The implementation is based on the compute unified device architecture (CUDA), a new parallel computing architecture introduced by NVIDIA [20].

The rest of this paper is organized as follows. The framework of the system is introduced in Sec. 2. The voxel model structure design oriented for parallel computing, details of the force/torque model, and the GPU-based parallel rendering techniques are given in Sec. 3. The simulation results and discussion are presented in Sec. 4. Finally, the paper concludes in Sec. 5.

## 2 System Framework

The system framework of our dental bone drilling simulator is shown in Fig. 1. The oral anatomy model was constructed directly from anonymized clinically indicated CBCT images, using the voxel-based method. The dimensions of the CBCT images used for the model construction are $416 \times 416 \times 256$. The corresponding spacing is $0.2\,\text{mm} \times 0.2\,\text{mm} \times 0.18\,\text{mm}$. In the preprocessing stage, the upper jaw and the maxillary teeth were segmented out and smoothed. The density of a voxel node was related to the intensity of a corresponding pixel in the smoothed image. The isosurface of the voxel model was constructed using the Marching Cubes (MC) algorithm [21]. The virtual drills and implants were also created and sampled in this stage. To start a typical training procedure for the dental implant surgery, a user first selects a surgical site, or a ROI (region of interest). Then, the user can choose the virtual drill, define its diameter and set the spindle speed of the hand-piece. During the real-time simulation, the virtual drill can be manipulated by the user through the stylus of Phantom Desktop, a 3DOF force feedback device by Sensable [22]. In the

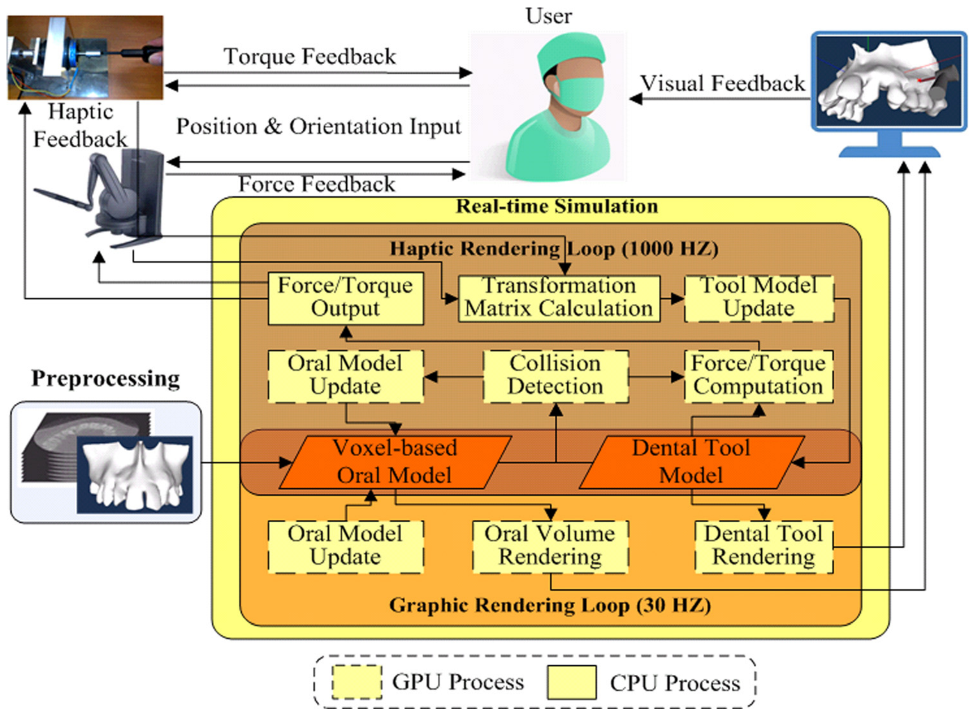


**Fig. 1 System framework**

haptic rendering loop (1000 Hz), a CPU process calculates the transformation matrix of the virtual drill based on the current position and orientation of the haptic stylus. Then, the transformation matrix is uploaded to the GPU. The collision detection, force computation, and the update of the oral and the tool models are all conducted on the GPU, based on the parallel computing architecture of the CUDA platform. The calculated feedback forces would be downloaded to the CPU and output to the haptic device. In the graphic rendering loop (30 Hz), the isosurface within the ROI is reconstructed on the GPU based on a parallel version of the MC algorithm, providing the visual feedback to the user. The system works in the same way during the implant inserting simulation, except that the virtual implant is controlled by a 1DOF torque feedback device developed in Ref. [23]. The torque resistance is computed by the torque model and output to the torque feedback device. Details of the proposed force/torque modeling approaches as well as the GPU-based haptic simulation based on CUDA architecture are presented in Sec. 3.

## 3 Haptic Simulation Based on GPU

Considering the current hardware computing power and real-time constraint for stable haptic rendering, force modeling based on penalty-based methods was applied in most of the previously developed haptic drilling systems. Most of these systems only targeted for a spherical-shaped drill bit due to its simplicity in collision detection and force calculation, although cylindrical or conical drill bits are commonly used in the real case. In addition, the traditional methods are not suitable for drilling applications as rotation-based drilling forces are not modeled properly. For a more precise drilling force model, more computations are required. Consequently, the computational efficiency becomes a much more severe issue. The modeling of the torque resistance is facing the same problem, where a more physically realistic oral model with higher resolution is required to build a practical training simulator for dentists to practice the procedures based on subtle force/torque variations. In this work, we tried to tackle this challenge through GPU-based parallel computing with the CUDA platform. More detailed description of the features and advantages of GPU and CUDA can be found in [24].

**3.1 GPU-Oriented Voxel Model Structure.** To describe our voxel model structure, two entities have to be defined first: Voxel Cell and Voxel Node. Voxel Cell, also termed as voxel, is a cubic volumetric element, which is a 3D counter-part of the pixel. Voxel Nodes are the eight vertices of the Voxel Cell. The relationship between these two entities is illustrated in Fig. 2. In our voxel model, only the voxel nodes are actually stored in the simulation system. A voxel cell can be represented by the voxel nodes located at its bottom-left-back corner and its top-right-front corner (nodes 0 and 6 in Fig. 2). The index of a voxel cell is defined to be the same as the index of its bottom-left-back voxel node. The density of voxel nodes may change during the run-time to simulate the material removal process. The connectivity and positions remain the same, assuming no deformation of the bone and tooth tissues.

Two types of data structures are generally used for voxel-based models: octrees and lookup tables (LUT). An Octree [25] partitions and organizes a 3D voxel space by recursively subdividing it into eight child nodes. The Octree data structure, once optimally

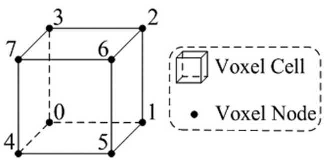


**Fig. 2 A voxel cell and its nodes**

built, is efficient for locating and neighbor searching of voxel nodes. However, there is a large amount of overhead in reconstructing the neighbor information in a dynamic environment. In addition, the recursive tree traversal process makes it difficult to be parallelized. On the other hand, with fixed voxel cell dimensions and a predefined resolution, neighbors of each voxel node stored in LUT can be obtained dynamically. More importantly, nodes stored in LUT are independent, which are suitable for the parallel computing on GPUs. Thus, an LUT is used to store voxel nodes in our approach.

To minimize the data transfers between CPU and GPU, which would incur great performance loss, all computations for isosurface reconstruction and force/torque modeling are carried out on the GPU. Considering the limited GPU memory, only a density value, a flag value and a collision indicator are recorded for each node. The data structure for each voxel node is defined as follows.

```
struct voxel_node {
    float density;
    bool flag;
};
```

The density value is assigned with the intensity value of a correlated pixel in CBCT images and is to be used for the force computation. Harder tissues have higher density value and vice versa. The flag value indicates whether or not a particular voxel is to be used for the isosurface reconstruction. During the real-time simulation, the density value is updated at haptic frequency (1000 Hz), while the flag value is updated at graphic frequency (30 Hz). The grid position of a voxel node is encoded in its index to LUT. Suppose the volume size is $2^x \times 2^y \times 2^z$, the grid position of voxel node $i$ can be extracted in run-time through a set of bit-wise operations as follows. The bitwise AND operator "&" takes two binary representations of two numbers and perform the logical AND operation on each pair of corresponding bits. The right shift operator "$\gg$" is equivalent to dividing $i$ by $2^x$ and by $2^{x+y}$, respectively.

$$\text{grid\_position.x} = i\ (2^x - 1) \tag{1}$$

$$\text{grid\_position.y} = (i \gg x)\ (2^y - 1) \tag{2}$$

$$\text{grid\_position.z} = (i \gg (x + y))\ (2^z - 1) \tag{3}$$

The stream processors on the GPU have fast bitwise operation units in their ALUs (Arithmetic Logic Unit). Besides, the computations for each voxel node are encapsulated in a lightweight CUDA thread, which is processed in parallel with negligible context switch overheads, so that the above computations can be quickly finished. It is a trivial task to calculate the Cartesian coordinate of each voxel node in run-time, once its grid position and the voxel cell dimensions (0.2 mm × 0.2 mm × 0.18 mm, the same as the CBCT image spacing used in this work) are known. The LUT for the voxel model is stored in the texture memory space on the GPU, which allows efficient memory access of the voxel data using a texture cache.

### 3.2 Haptic Modeling and GPU-based Implementation

*3.2.1 Force Modeling.* Our force model was inspired by the VPS model. However, unlike the VPS model, which represents dynamic objects by a set of points sampled from the entire surface, our model only collects point samples along the cutting edges of the drill bit. These sampled points are called cutting elements. During each haptic update, the elementary forces on each cutting element are calculated based on its current position in the oral anatomy, which is a combination of the hand-piece's global motion transformation and the cutting element's rotational motion with respect to the drill axis. The force model illustration is shown in Fig. 3. The normal elementary force $F_i^n$ and the tangential

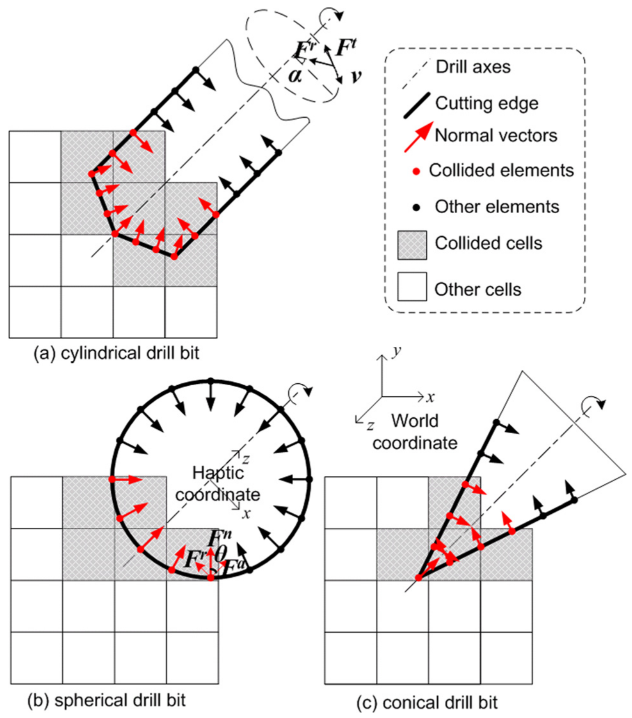


Fig. 3 Illustration of force model

elementary force $F_i^t$ acting on cutting element $i$ are calculated as follows.

$$F_i^n = \Delta_i k_i^n D_i \quad (4)$$

$$F_i^t = \Delta_i k_i^t D_i \quad (5)$$

$\Delta_i$ equals 1 if element $i$ collides with a voxel cell and equals 0 otherwise; $k_i^n$ and $k_i^t$ are the stiffness values in the normal and tangential directions, respectively (Presently, the $k$ values were adjusted according to users' feedback on the trial simulator. They would be calibrated through the measurement of actual drilling forces in the future); $D_i$ is the approximated density of the oral tissue where the element is currently located.

If $\theta_i$ is defined as the angle between the normal vector and the drill axis, the elementary force in the axial and the radial directions can be calculated as:

$$F_i^a = F_i^n \cos\theta_i \quad (6)$$

$$F_i^r = F_i^n \sin\theta_i \quad (7)$$

To render forces on a haptic device, $F_i^a$ and $F_i^t$ should be projected to the coordinate of the haptic stylus (haptic coordinate). Suppose at time $t$, element $i$ has moved $\alpha$ degrees with the rotation of the drill bit, the following transformations are applied:

$$F_i^x = -F_i^r \cos\alpha - F_i^t \sin\alpha = -F_i^r \cos\omega t - F_i^t \sin\omega t \quad (8)$$

$$F_i^y = -F_i^r \sin\alpha + F_i^t \cos\alpha = -F_i^r \sin\omega t + F_i^t \cos\omega t \quad (9)$$

$$F_i^z = F_i^a \quad (10)$$

where $\omega$ is the spindle speed of the handpiece.

The final force output is obtained by summing all the elementary forces:

$$F^x = \sum_i \left(-F_i^n \sin\theta_i \cos\omega t - F_i^t \sin\omega t\right) \quad (11)$$

$$F^y = \sum_i \left(-F_i^n \sin\theta_i \sin\omega t + F_i^t \cos\omega t\right) \quad (12)$$

$$F^z = \sum_i F_i^n \cos\theta_i \quad (13)$$

*3.2.2 Torque Modeling*. For the surgical screw insertion process, there are normally three major phases.

(1) Insertion phase: At the beginning of the screw insertion, the resistance torque rises gradually due to the increasing friction between the bone tissue and the screw.
(2) Tightening phase: After the screw head touches the bone surface, the resistance torque rises sharply, usually over one rotation, as the screw threads are forced against the newly formed bone threads.
(3) Stripping phase: If the rotation continues after the tightening phase, the resistance torque would drop rapidly, because the bone threads would be removed/damaged by the overturn of the screw.

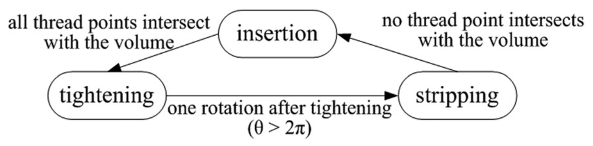
Successful insertion of the dental implant would only include the insertion phase and the tightening phase. When the stripping phase occurs, the procedure fails due to the loosening of the implant. The torque model should be able to simulate all three phases so that the user can practice achieving the maximum implant-bone tightness in the tightening phase and avoiding the stripping phase. To simulate all three torque phases with patient-specific tissue properties, a voxel-based torque computation model has been developed. The model calculates the torque response based on the geometry of the screw implant and the property of the contacted oral tissue. More specifically, it calculates the torque contribution of each thread element based on the collision detection results. The magnitude of the individual torque contribution, or elementary torque, is related to the density of the collided oral voxel and the current torque mode (insertion/tightening/stripping). Summation of all of the elementary torques will give the total torque response at the current moment. The details of the torque computation model are described as follows.

The state transition of the three torque modes is shown in Fig. 4. The torque modes are switched according to the rotary position of the implant and the collision detection results. The torque mode is initially set to insertion mode. It would switch to tightening mode when all the thread elements intersect with the voxel-based oral model. In other words, the full immersion of the screw thread into the oral volume will trigger the transition from insertion mode to tightening mode. Stripping mode starts after one rotation of the implant in the tightening mode. If none of the thread elements intersects with the voxel model after stripping, meaning that the implant is no longer contacting the oral volume, the torque mode would be reset to insertion mode again. The illustration of the torque model is shown in Fig. 5.

The elementary torque $T_i$ contributed by thread element $i$ can be calculated as follows:

$$T_i = F_i r_i = \Delta_i K D_i r_i \quad (14)$$

$F_i$ is the tangential elementary force on thread element $i$. $r_i$ is its radial distance to the screw axis. Definitions of $\Delta_i$ and $D_i$ are the same as those in Eqs. (4) and (5). $K$ is the virtual contact stiffness which is related to the current torque mode.



all thread points intersect with the volume
insertion
no thread point intersects with the volume
tightening
one rotation after tightening (θ > 2π)
stripping

Fig. 4 State transitions of torque modes

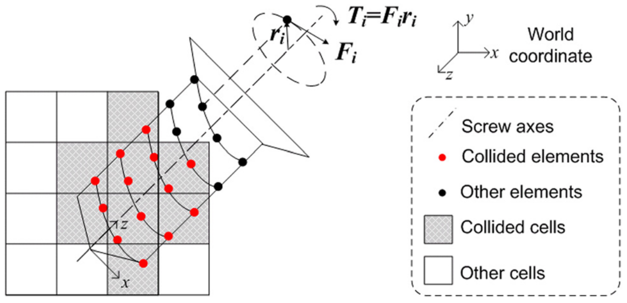


**Fig. 5 Illustration of torque model**

For insertion mode

$$K = K^{\text{ins}} \tag{15}$$

For tightening mode

$$K = K^{\text{ins}}(1 + s\theta) \tag{16}$$

For stripping mode

$$K = K^{\text{ins}}(1 + 2\pi s) \tag{17}$$

$K^{\text{ins}}$ is a constant representing the virtual contact stiffness for the insertion mode. $s$ is a constant scaling factor. $\theta$ records the rotation angle of the screw from the beginning of the tightening phase ($0 \leq \theta \leq 2\pi$). In other words, $K$ increases from $K_{\text{ins}}$ to $K_{\text{ins}}(1 + 2\pi s)$ during the tightening phase.

The total resistance torque is the sum of all elementary torques, as shown in the equation below

$$T = \sum_i^n T_i = \sum_i^n \Delta_i K D_i r_i \tag{18}$$

$n$ is the number of thread elements. The voxel node density remains the same for the insertion and tightening phase, as it is assumed that there is no material loss during these two phases. As for the stripping phase, $D_i$ is decreased in each haptic loop to simulate the damage of bone thread. (Currently, parameter $K_{\text{ins}}$ and $s$ were adjusted according to users' feedback on the trial simulator. They would be calibrated through the measurement of actual resistance torque in the future).

*3.2.3 GPU-Based Parallel Implementation.* If the number of sampled elements is small, each element will contribute significantly to the final force output. Consequently, there can be sharp changes in the feedback force/torque, when an element enters or leaves a voxel cell, or when it travels through the boundary of different tissue layers. Due to its limited stiffness, the haptic device can be unstable with abrupt force/torque variations. Increasing the element number will mitigate the above effect. As traditional methods conduct collision detection and force/torque computations element by element, increasing the element number will increase the computation time. To solve this problem, GPU-based parallel computing is introduced.

The cutting elements are stored in a lookup table called cutLUT. Each entry of the cutLUT stores a pointer to the CuttingElem data structure, which holds the initial position of a cutting element, its normal vector, as well as the angle between the normal vector and the drill axis.

```
struct CuttingElem {
    float init_pos [3];
    float normal [9];
    float theta;
};
```

The thread elements are stored with a lookup table called threadLUT. Each entry of the threadLUT contains a pointer to the ThreadElem data structure, which holds the initial position of a thread element and its radial distance to the screw axis.

```
struct ThreadElem {
    float init_pos [3];
    float radial_distance;
};
```

Both the cutLUT and the threadLUT are stored in the constant memory space on the GPU, since it is constant during the runtime. The access of the constant memory is even faster than that of the texture memory, due to the constant memory cache [20].

Based on the proposed force/torque model, the kernel function on the GPU for force rendering is divided into nine device functions, as shown in Fig. 6.

*updateElemPos*() calculates the current Cartesian coordinates of an element using the drill/implant transformation matrix (stored in the Pinned Memory of the GPU) and the original position of the element in cutLUT/threadLUT (stored in the Constant Memory of the GPU). The Pinned Memory is nonpageable and allows direct read/write of CPU memory by CUDA threads. *getCollidedVoxelCell*() returns the index of the voxel cell that a sampled element collided with. As we aligned the voxel space with the axes of the world coordinate system, collision detection is simply performed by:

```
if ((elem_pos.x > cell_min.x)&&(elem_pos.x <= cell_max.x)&&
  (elem_pos.y > cell_min.y)&&(elem_pos.y <= cell_max.y)&&
  (elem_pos.z > cell_min.z)&&(elem_pos.z <= cell_max.z))
    return true;
else
    return false;
```

elem_pos is the Cartesian coordinate of the element returned by *updateElemPos*(). cell_min and cell_max are the coordinates of nodes 0 and 6 of a target voxel cell.

Although the above algorithm is quite fast for a single voxel cell, given millions of voxel cells in the workspace, the whole collision detection process is still time-consuming. Instead of the traditional voxel-by-voxel collision detection, a top-down approach is proposed here, which is inspired by the Octree data structure. The collision detection process using this approach is illustrated in Fig. 7.

Suppose the index of the collided cell is $i$, then *getVoxelNodes*() computes the indices of its eight voxel nodes as follows:

```
indices[0] = i;
indices[1] = indices[0] + 1;
indices[2] = indices[1] + volume_size.x;
indices[3] = indices[2] − 1;
indices[4] = indices[0] + volume_size.y × volume_size.x;
indices[5] = indices[4] + 1;
indices[6] = indices[5] + volume_size.x;
indices[7] = indices[6] − 1;
```

The distances between the cutting element and the eight voxel nodes are computed by *getNearestVoxelNode*(). The density of the nearest voxel node will be decreased by *updateDensity*() to simulate the cutting process. *getInterpDensity*() calculates $D_i$ in Eqs. (4) and (5) and Eq. (14), through trilinear-interpolation from the eight neighboring voxel nodes for improved precision

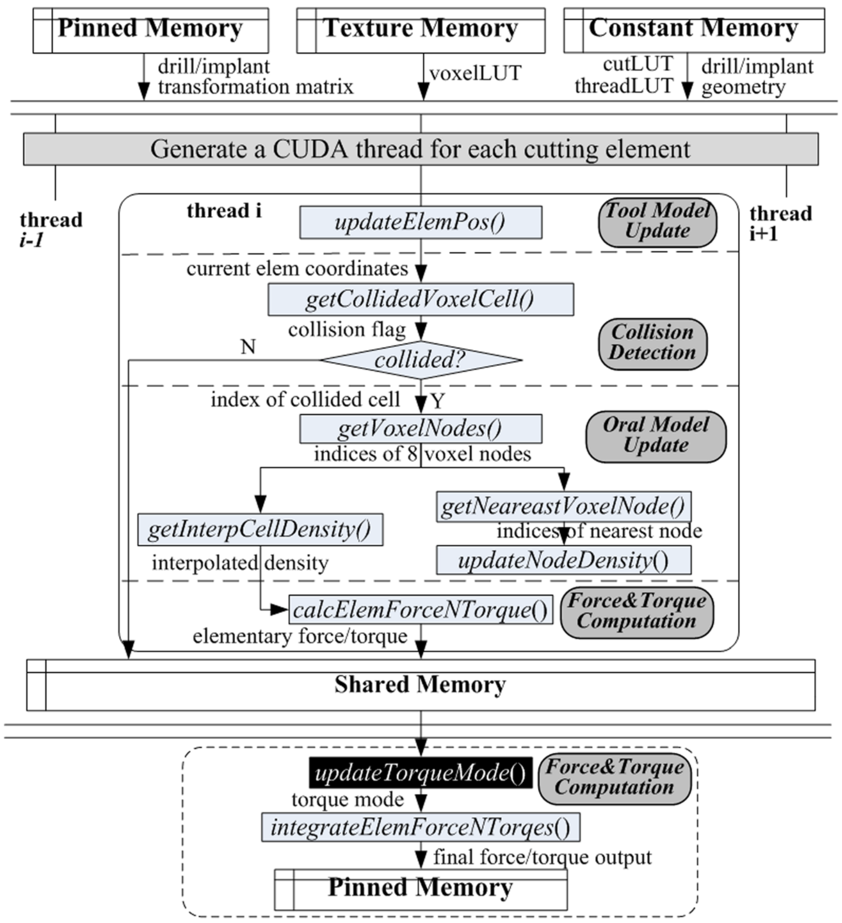


Fig. 6 Flowchart of the force computation kernel

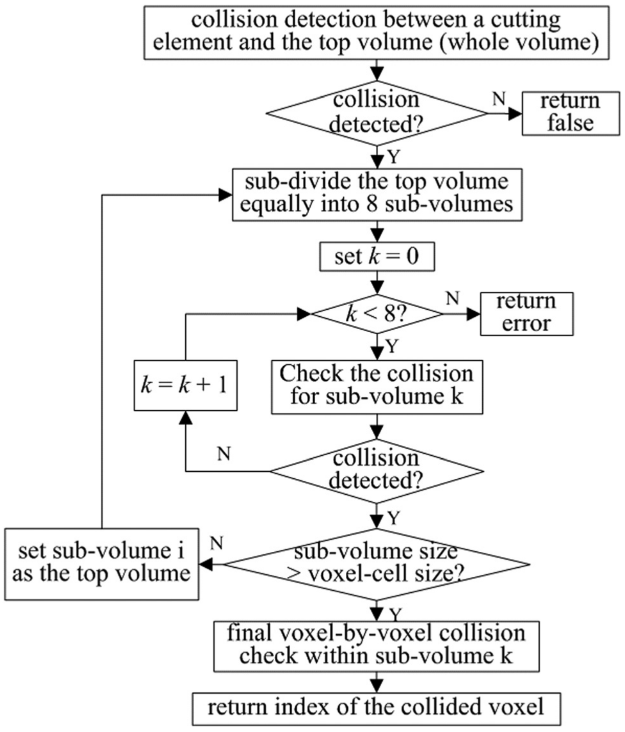

collision detection between a cutting element and the top volume (whole volume)
collision detected?
N
return false
Y
sub-divide the top volume equally into 8 sub-volumes
set $k = 0$
$k < 8$?
N
return error
Y
Check the collision for sub-volume k
$k = k + 1$
N
collision detected?
Y
set sub-volume i as the top volume
N
sub-volume size > voxel-cell size?
Y
final voxel-by-voxel collision check within sub-volume k
return index of the collided voxel

Fig. 7 The top-down collision detection algorithm

$$\begin{aligned}D_i = {} & D_0(1-x_i)(1-y_i)(1-z_i) + D_1x_i(1-y_i)(1-z_i) \\ & + D_2x_i(1-y_i)z_i + D_3(1-x_i)(1-y_i)z_i \\ & + D_4(1-x_i)y_i(1-z_i) + D_5x_iy_i(1-z_i) \\ & + D_6x_iy_iz_i + D_7(1-x_i)y_iz_i \end{aligned} \tag{19}$$

$D_0 - D_7$ are the densities of the eight neighboring voxel nodes, $x_i$, $y_i$, $z_i$ is the current world coordinate of element $i$. The trilinear interpolation can be conducted in a very efficient manner using 3D texture memory under CUDA.

*calcElemForceNTorque*() computes the contribution of a sampled element to the net force/torque (based on Eqs. (8)–(10) and Eqs. (14)–(17)). The calculated elementary forces/torques are stored in shared memory, which allows for fast data access within the same thread block. *updateTorqueMode*() is called only in torque computation, which changes the torque mode according to Fig. 4. *integrateElemForceNTorque*() accumulates the elementary forces/torques in the shared memory to get a final force/torque output. The integration process is conducted with a commonly used parallel computing pattern called reduction (or reduction tree), which is illustrated in Fig. 8.

**3.3 Isosurface Updating.** The isosurface of the oral volume needs to be updated in real time to provide a visual feedback of the drilling effect. The original marching cubes algorithm has been adapted for parallel implementation on the GPU. During the real-time simulation, a CUDA thread is generated for each voxel node at 30 Hz frequency, as required by real-time constraint for graphic rendering. Each thread would check the density of a given node, and clear its flag if the density is decreased to 0. Then, based

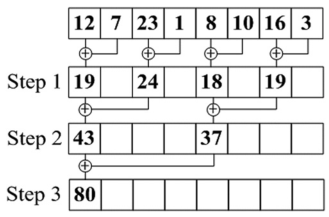


**Fig. 8 Force integration based on reduction tree**

on the updated flag, the triangle vertices for each boundary voxel cell are located and normals are computed. Since threads are running simultaneously, the process for isosurface reconstruction is very efficient. Moreover, because the generated triangles are directly written into GPU memory, the graphic rendering is also quite fast.

## 4 Results and Discussion

To evaluate the proposed methods, a trial simulation experiment was performed using the proposed simulator. CBCT images of a patient with a missing molar were used for the oral model construction. A drilling site (or ROI) was selected first on the isosurface of the voxel-based model with an axis-aligned bounding box (Fig. 9(*a*)). Voxel nodes within ROI would be used for haptic rendering. During the real-time haptic simulation, the densities of the collided voxel nodes were updated, and isosurface in the ROI was reconstructed at about 30 Hz (Fig. 9(*b*)). The trial surgery started by performing a pilot drilling procedure at the hole in the maxilla (caused by the missing molar). The drilling was stopped after the penetration of the maxillary sinus (Fig. 9(*c*)). It should be noted that in the real case, experienced dentists stop drilling once they sense that the drill is in contact with the maxillary sinus, to prevent irreversible damage to this critical structure. However, in this trial procedure, the drill continues to penetrate the maxillary sinus after the contact in order to test the response of the proposed force model at different tissue layers.

Figure 10 illustrates the real-time rendered drilling forces in $x$, $y$ and $z$ dimensions ($F^x$, $F^y$ and $F^z$ in Eqs. (11)–(13)) during the procedure. These forces were output to the haptic device (Phantom Desktop in our system) in 1000 Hz. In this trial procedure, the implant drill passed through a hard cortical bone tissue (Region A),

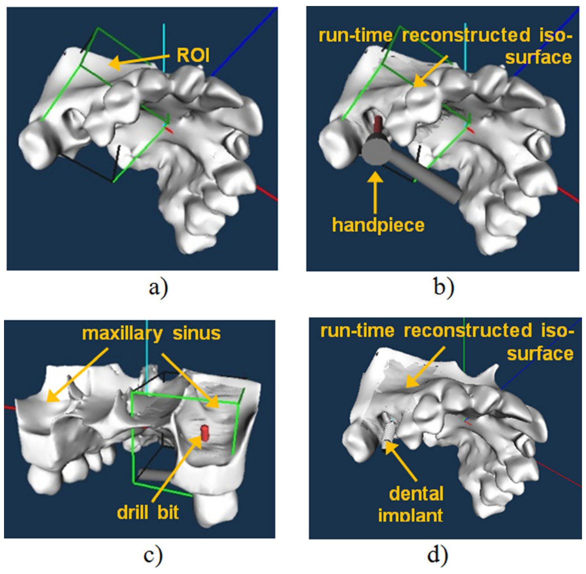

ROI
run-time reconstructed iso-surface
handpiece
a)
b)
maxillary sinus
drill bit
run-time reconstructed iso-surface
dental implant
c)
d)

**Fig. 9 Graphic rendering results: (*a*) ROI selection; (*b*) real-time interaction during drilling; (*c*) penetration of the maxillary sinus; and (*d*) real-time interaction during implant insertion**

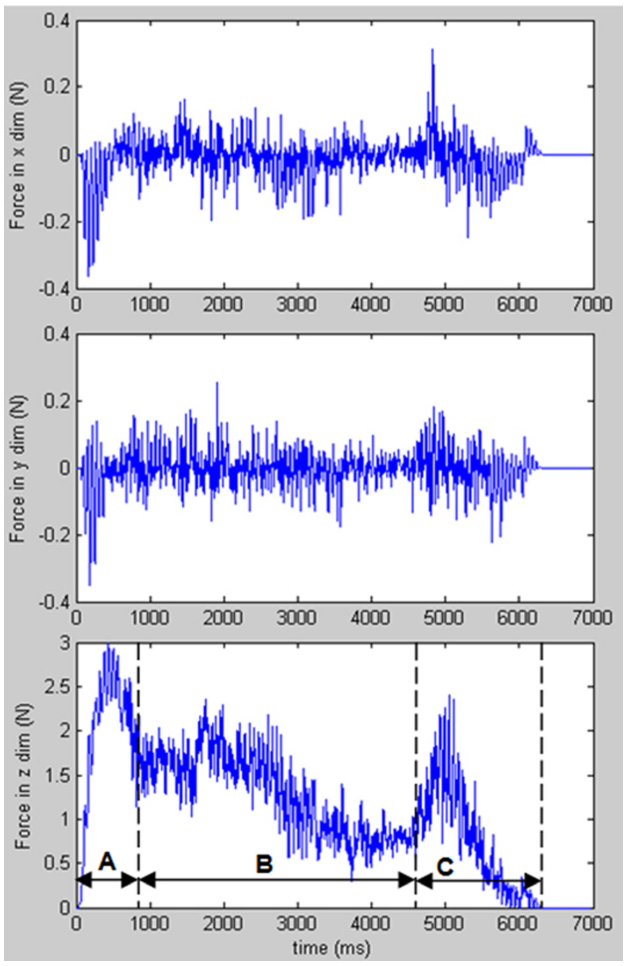


**Fig. 10 Haptic rendering results of drilling forces**

a softer, spongious cancellous bone tissue (Region B), and stopped after the penetration of the maxillary sinus (Region C). The relevant haptic properties of the involved tissues can be represented using the proposed force model, as shown in Fig. 10. The force in the $z$ dimension simulated the thrust force. A sudden increase of thrust force could be seen at the first contact of the cortical bone, followed by smaller resistance at the cancellous bone. Finally, an abrupt rise of thrust force signaled the hitting of the maxillary sinus. In addition, the oscillation of the implant drill could be simulated by forces rendered in $x$ and $y$ dimensions. When all cutting elements are cutting at the same tissue, the magnitudes of the radial force ($F_i^r$) and the tangential force ($F_i^t$) acting on an element are almost the same as the ones on its symmetric counterpart (with respect to the drill axis), but the corresponding force directions are opposite. Thus, in this case, the oscillation forces are quite small. However, at the beginning stage of drilling, the oscillation forces are relatively large, as shown in Region A of Fig. 10. To explain this phenomenon, consider at a particular moment of this stage: a cutting element was cutting a hard cortical bone, while its symmetric counterpart was cutting in the air. In this situation, $F_i^r$ and $F_i^t$ contributed by this pair differ greatly, resulting in large oscillation. This is consistent with drilling oscillation in the real case. Due to the same reasons, large oscillation also occurs during the penetration at the boundary of different tissues.

To evaluate the torque model and the torque feedback device, a trial procedure for the dental implant insertion was also performed. Figure 9(*d*) shows the real-time graphic rendering results when inserting a dental implant into the pilot hole. In order to test the torque response of the simulator in all three phases, screwing continued until nearly no torque can be felt.

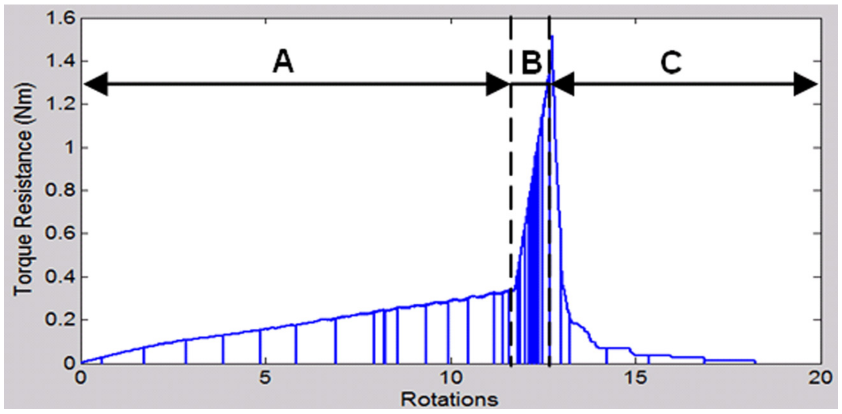


**Fig. 11 Haptic rendering results of resistance torques**

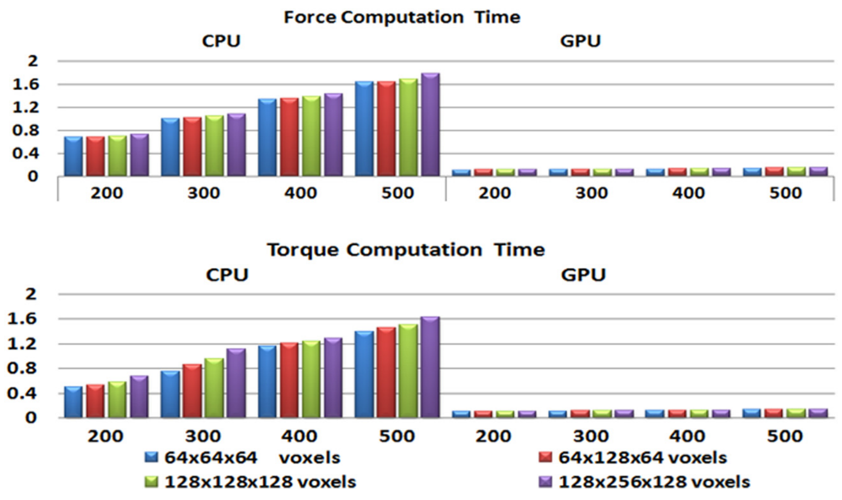


**Fig. 12 Performance comparison between CPU and GPU**

The recorded torque-rotation profile during this trial procedure is shown in Fig. 11. The three major phases during the implant insertion can be easily identified from the above figure. During the insertion phase (Region A), the screw implant was moving toward the oral model along its axis. The resistance torque gradually increased as more and more thread points came into contact with the bone voxels, contributing their elementary torques to the total output. The tightening phase (Region B) began once the screw head touched the bone surface. The screw was rotating at the same position because its advancing movement was stopped by the screw head. The number of thread points intersecting with the bone volume was kept the same during this phase. The sudden rise of the resistance torque was caused by the rapid increase of the virtual contact stiffness $K$ in Eq. (14), indicating growing load between the screw threads and newly formed bone threads. After one rotation, it finally came to the stripping phase (Region C). In this phase, the density of the contacted bone voxels was decreasing to simulate the damage of the newly formed bone threads. As a result, a quick drop of resistance torque could be felt. It should be noted that there were a few dips in the torque-rotation profile. These dips were generated when the user stopped screwing momentarily during the procedure, resulting in zero torque resistance at a given moment. The dips were more intensive in the tightening phase, as the rotation of the screwdriver was slowed down and the user stopped more frequently due to larger torque resistance. The torque-rotation profile is consistent with the profiles given in Refs. [19,26].

To compare the computational performance on the CPU-based environment and the GPU-based environment, the same force/torque model was implemented in two versions: a serial version on the CPU side and a parallel version on the GPU side. Both versions are programmed in C++ with Microsoft Visual Studio 2003 and are compiled with default compilation flags.The force/torque computation time for each haptic cycle was recorded during the trial procedures with fixed drilling location and direction. Considering the variable response time at every cycle and the related implementation overheads, an averaged cycle time for a sample haptic drilling was used for performance comparison. The test was conducted on a Dell Precision 380 equipped with a 64-bit Intel® Pentium® D CPU (3.20 GHz & $2 \times 1$ MB L2 cache), an 800 MHz Front side bus, a 4 GB DDR2 533 MHz memory, an NVIDIA Quadro FX 3800 graphics card (1 GB GDDR3, 256-bit, 51.2 GB/s, CUDA compute capability 1.3) and the Windows XP operating system.

Figure 12 gives a summarized view of the measured cycle times with different volume sizes and different numbers of cutting elements. It can be seen that the force/torque computation time increases very slowly with the increase of the volume size, thanks to the Octree-inspired top-down collision detection method described in Sec. 3.2.3. However, for the CPU-based serial implementation, a drastic rise of time cost can be seen with a larger number of cutting elements. The desired haptic frame rate (1000 Hz or 1 ms) cannot be achieved with more than 300 cutting elements. Actually, based on our experience, the force/torque feedback becomes unstable when using 300 elements with a $128 \times 128 \times 128$ volume, probably due to the extra time cost and overhead for the marching cubes isosurface reconstruction. In contrast, for the CUDA-parallelized implementation on the GPU, a remarkable speedup of 6–12 times can be gained. The computation time is still under 0.2 ms even with a volume size of $128 \times 256 \times 128$ and 500 cutting elements, which can satisfy the real-time haptic rendering constraint.

## 5 Conclusions and Future Work

This paper presents the development of a haptics-based training simulator for dental implant surgery. For the accurate representation of the oral anatomy, patient-specific CBCT images are utilized to construct the oral model, using the voxel-based approach. The core, and also the major advantage of our training simulator is the adapted VPS drilling force model, the voxel-based torque model and the GPU-based parallel architecture. In the proposed force model, cutting elements are sampled from the cutting edges of a drill. The drilling forces are computed based on the rotational motion of the sampled cutting elements. The method captures the essential features of the drilling process. Appropriate thrust force and oscillation can be simulated during real-time simulation, as demonstrated in the trial results. With the simulated drilling force feedback, users can become familiar with the differences in tactile sensation when drilling through different oral tissues, avoiding severe damage to critical dental tissues and structures. Moreover, with the simulated oscillation forces, users can learn to control the drill oscillation after proper training. In the proposed torque model, thread elements are sampled along the screw thread. The resistance torque is modeled based on the voxel density and the virtual contact stiffness throughout the insertion, tightening and stripping phases. With the simulated resistance torque feedback, users are able to learn what stripping feels like and how much torque may cause stripping, in order to prevent stripping while trying to achieve the maximum tightness between the dental implant and the bone tissue. The real-time constraint for haptics-based rendering has been achieved by parallelizing the force/torque computation algorithm on the GPU, using the NVIDIA CUDA architecture. The CPU-GPU comparison results show an impressive speedup with the GPU-based method, which makes it possible to simulate larger volumes and greater numbers of elements, if necessary.

Future work will measure the actual response force/torque when performing the pilot drilling procedure and the implant insertion procedure on the jaw of an adult pig. The collected force/torque data would be used to calibrate the force/torque model, thus providing better accuracy for the simulator. Additionally, the system can be enhanced by simulating the gum tissue, bone debris accumulation, irrigation and suction while drilling, as well as the mounting of an artificial crown.

## References

[1] http://www.denx.com/DentSim/overview.html
[2] http://www.kavousa.com/Default.aspx?navid=5210&oid=009&lid=Us
[3] Wang, D., Yuru, Z., Yuhui, W., Lee, Y. S., L. Peijun, and Yong, W., 2005, “Cutting on Triangle Mesh: Local Model-Based Haptic Display for Dental Preparation Surgery Simulation,” IEEE Trans. Vis. Comput. Graph., **11**, pp. 671–683.
[4] Rhienmora, P., Haddawy, P., Dailey, M. N., Khanal, P., and Suebnukarn, S., 2008, “Development of a Dental Skills Training Simulator Using Virtual Reality and Haptic Device,” NECTEC Tech. J., **8**, pp. 140–147.
[5] Cho, J. H., Jung, H., Yu, I., Lee, K., Lee, D. Y., Ahn, H. S., Park, I., Yeo, S. H., and Han, S.-H., 2007, “Surface-Data-Based Haptic Rendering for Simulation of Surgery of Closed Reduction and Internal Fixation,” Lyon, France, pp. 210–213.
[6] Esen, H., Yano, K. I., and Buss, M., 2008, “Bone Drilling Medical Training System,” *Springer Tracts in Advanced Robotics*, Vol. 45, STAR, pp. 245–278.
[7] Luciano, C., Banerjee, P., and DeFenti, T., 2009, “Haptics-Based Virtual Reality Periodontal Training Simulator,” Virtual Reality, **13**, pp. 69–85.
[8] Heiland, M., Petersik, A., Pflesser, B., Tiede, U., Schmelzle, R., Höhne, K. H., and Handels, H., 2004, “Realistic Haptic Interaction for Computer Simulation of Dental Surgery,” Int. Congr. Ser., **1268**, pp. 1226–1229.
[9] Wu, J., Yu, G., Wang, D., Zhang, Y., and Wang, C. C. L., “Voxel-Based Interactive Haptic Simulation of Dental Drilling,” San Diego, CA, pp. 39–48.
[10] Yau, H. T., Tsou, L. S., and Tsai, M. J., 2006, “Octree-Based Virtual Dental Training System With a Haptic Device,” Comput.-Aided Des. Appl., **3**, pp. 415–424.
[11] Kim, K., Park, Y.-S., and Park, J., 2008, “Volume-Based Haptic Model for Bone-Drilling,” Piscataway, NJ, pp. 255–259.
[12] He, X., and Chen, Y., 2006, “Bone Drilling Simulation Based on Six Degree-of-Freedom Haptic Rendering,” Euro-Haptics, Paris, France, pp. 203–210.
[13] Acosta, E., and Liu, A., 2007, “Real-Time Volumetric Haptic and Visual Burrhole Simulation,” Charlotte, NC, pp. 247–250.
[14] Morris, D., Sewell, C., Barbagli, F., Salisbury, K., Blevins, N. H., and Girod, S., 2006, “Visuohaptic Simulation of Bone Surgery for Training and Evaluation,” IEEE Comput. Graph. Appl., **26**, pp. 48–57.
[15] Agus, M., Giachetti, A., Gobbetti, E., Zanetti, G., and Zorcollo, A., 2003, “Real-Time Haptic and Visual Simulation of Bone Dissection,” Presence, **12**, pp. 110–122.
[16] Kim, L., and Park, S. H., 2006, “Haptic Interaction and Volume Modeling Techniques for Realistic Dental Simulation,” Vis. Comput., **22**, pp. 90–98.
[17] McNeely, W. A., Puterbaugh, K. D., and Troy, J. J., 1999, “Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling,” New York, NY, pp. 401–408.
[18] Renz, M., Preusche, C., Pötke, M., Kriegel, H.-P., and Hirzinger, G., 2001, “Stable Haptic Interaction With Virtual Environments Using an Adapted Voxmap-Pointshell Algorithm,” Eurohaptics, pp. 149–154.
[19] Majewicz, A., Glasser, J., Bauer, R., Belkoff, S. M., Mears, S. C., and Okamura, A. M., 2010, “Design of a Haptic Simulator for Osteosynthesis Screw Insertion,” 2010 IEEE Haptics Symposium (Formerly Known as Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems), March 25–26, Piscataway, NJ, pp. 497–500.
[20] Nvidia, 2010, “NVIDIA CUDA Programming Guide, Version 3.0.”
[21] Lorensen, W. E., and Cline, H. E., 1987, “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” Comput. Graph. (ACM), **21**, pp. 163–169.
[22] http://www.sensable.com/haptic-phantom-desktop.htm
[23] Zheng, F., Lu, W. F., and Wong, Y. S., 2011, “Voxel-Based Haptic Training Simulator for Screw Insertion in Knee Osteotomy,” Automation, Robotics and Applications (ICARA), pp. 179–183.
[24] Zheng, F., Lu, W. F., Wong, Y. S., and Foong, K. W. C., 2011, “GPU-Based Haptic Simulator for Dental Bone Drilling,” ASME Conference Proceedings, pp. 1419–1428.
[25] Meagher, D., 1982, “Geometric Modeling Using Octree Encoding,” Comput. Graph. and Image Process., **19**(2), pp. 129–147.
[26] Thomas, R. L., Bouazza-Marouf, K., and Taylor, G. J., 2008, “Automated Surgical Screwdriver: Automated Screw Placement,” Proc. Inst. Mech. Eng., Part H: J. Eng. Med., **222**, pp. 817–827.