

# Real-time volumetric haptic and visual burrhole simulation

Eric Acosta\*, Alan Liu

National Capital Area Medical Simulation Center  
Uniformed Services University

## ABSTRACT

This paper describes real-time volumetric haptic and visual algorithms developed to simulate burrhole creation for a Virtual Reality-based craniotomy surgical simulator. A modified Voxmap point-shell algorithm [4, 7] is created to simulate haptic interactions between bone cutting tools and voxel-based bone. New surface boundary detection and force feedback calculation methods help reduce “force discontinuities” of the original Voxmap point-shell algorithm. To maintain stable haptic update rates, new forces are calculated outside the haptics rendering loop. A multi-rate haptic solution [2] is used to introduce calculated forces into the haptics loop and to interpolate forces between updates. A bone erosion method is also created to simulate bone drilling capabilities of different tools. 3D texture-based volume rendering is used to display the bone and to visually remove bone material due to drilling in real-time. Volumetric shading is computed by the GPU of the video card. The algorithms described make it possible to simulate several tools typically used for a craniotomy. Realistic 3D models are also created from real surgical tools and controlled by the haptic device.

**Keywords:** Volumetric haptics, volume rendering, surgical simulation, bone drilling.

**Index Terms:** I.3.6 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques

## 1 INTRODUCTION

Head trauma can cause injuries to the brain that lead to complications and death if left untreated. Surgery is required for treatment when conservative measures are ineffective. To assist with surgical training, we are developing a Virtual Reality-based medical training simulator to address internal bleeding of the brain (intracranial hematoma). Access to the brain requires a section of bone to be removed from the skull with a procedure called a craniotomy. An important step in this procedure involves cutting burrholes in the skull using powered tools. In order to practice this skill within a simulator, real-time haptic, bone erosion, and visual algorithms are required to simulate surgical tools for creating burrholes. This paper describes the algorithms created for this purpose.

The described volumetric haptic work is based on the Voxmap point-shell algorithm described by McNeely et al. [4] and later by Renz et al. [7]. The algorithm represents a virtual environment (VE) using a spatial occupancy map called a voxmap. Objects are encoded into the voxmap using voxels that identify which locations in space are occupied. The haptic device is represented as a set of points that lie along the surface of its virtual representation. Collisions between the haptic device and the VE are detected by probing the voxmap with the points of the point-shell, Figure 1. A force response is then calculated for the haptic device based on the voxel-point intersections. However, the force calculation method

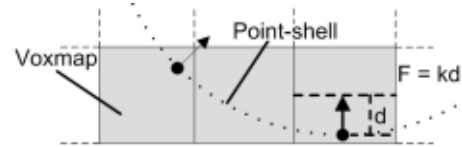


Figure 1: Point-shell interacting with voxmap.

in [4] and [7] introduces force discontinuities at voxel boundaries. In [7] they described a method to help reduce these “force distractions,” but still rely on a “virtual coupling” scheme to stabilize haptic interactions. Petersik et al. described a sub-voxel method to better approximate the surface boundaries [6]. Unfortunately, the sub-voxel method is computationally very expensive during volume updates and requires another simple force calculation method while drilling. We address the haptic interaction instability of the Voxmap point-shell algorithm by modifying its surface boundary detection and force calculation methods. The “virtual coupler” is not required. Also, the same force calculation method is used regardless if drilling or not.

To model bone drilling, Morris et al. [5] use the same point-shell points to compute haptic interactions and bone erosion for spherical drill bits. The amount of bone removed is based on a “drilling power” assigned to points and on the points’ distance to the instrument’s main axis. Agus et al. [1] developed a model of bone erosion as a function of drilling force and angular velocity. They sample the bone data using a small region located within a spherical drill bit to compute force feedback and bone removal. Mixing the voxel sampling methods for haptics and bone erosion works well for spherical drill bits since all areas can cut bone. However, modeling different types of tool bits (e.g., perforator bit shown in Figure 3) requires a method to easily define areas that can/cannot remove bone and/or remove bone at varied rates.

Advancements in graphics rendering technologies make it possible to leverage texture-based rendering capabilities to display volumetric datasets in real-time [3]. However, the basic texture-based volume rendering approach does not include shading. Previous work on volumetric shading for texture-based rendering methods has been performed [3]. We also utilize the GPU to compute volumetric shading. We also show how texture-based rendering can be used to remove bone material in real-time while drilling.

Several virtual bone cutting tools have been simulated based on the work described in this paper. Figure 2 shows example bone drills and perforator tools that were modeled after real surgical instruments. These tools are controlled by a PHANTOM haptic device that provides force feedback to the user. A haptic workbench [9] is used to generate the VE with 3D stereoscopic visual and haptic feedback. The haptic workbench allows the visual and haptic workspaces to be co-registered so that both visual and haptic feedback appear to originate from the same location.

## 2 HAPTIC RENDERING

Two main tasks of haptic rendering involve 1) detecting when a tool bit collides with the virtual bone material and 2) responding

\*e-mail: eacosta@simcen.usuhs.mil; http://simcen.usuhs.mil



Figure 2: Simulated surgical tools typically used in clinical practice for a craniotomy. From left to right: Bone drill and perforator from Medtronic, and bone drill and perforator from Stryker.

to a collision by calculating a force feedback for the haptic device. The bone material is represented with the voxels that make up the voxmap. Each voxel encodes bone density, density gradient (unit vector), and color. The tip of the haptic device is represented in the VE as a point-shell, similar to Figure 3. Each haptic point stores its position relative to the tool bit's center and an inward pointing tool vector that is used for calculating the collision force. The spatial positioning of the haptic points impacts the calculation of the force [7]. To ensure an even and symmetric point distribution, the positions and tool vectors of the haptic points are computed based on either a spherical or a cylindrical approximation of the tool bits. Several parameters (e.g., radius, height, number stacks and slices, and voxel size) can be specified in order to automatically generate the point-shell for bits of different shapes and sizes. The parameters also help control the number of haptic points that are generated.

The location and tool vector orientation of the haptic points are transformed based on the position and orientation of the haptic device. Collisions between the tool bit and bone material are checked by sampling the voxmap at the transformed locations of the haptic points. A collision is detected when a point intersects a voxel having a non-zero density value.

A force is computed for each colliding haptic point based on Hooke's Law  $F = kd$ , where  $d$  is a point's penetration depth with the bone material surface and  $k$  is a stiffness constant for the object. To compute  $F$ , a surface voxel needs to be located. In [4, 7] the Voxmap's voxels are labeled as interior, surface, or free. A surface detection algorithm would be required in our case to dynamically update the status of voxels as bone material is removed while drilling. To avoid this, the tool vector is followed until an empty voxel is found. However, simply following the tool vector to the



Figure 3: Cylindrical and spherical point-shell approximations for a perforator bit (316 haptic points) and a round bit (72 haptic points).

surface introduced problems. For example, the length of a point's vector can be followed just under the surface without finding an empty voxel. Also, large force magnitude fluctuations can result if the locations of consecutively identified surface voxels changes significantly. To address these issues, our method utilizes the gradient information of voxels to help locate the surface. The tool vector is followed at voxel-sized intervals until a voxel with a non-zero gradient is found. The gradient is then followed to the surface. In most cases, the surface is located within a few samples using the gradient since the bone material is stiff and little penetration is allowed. This helps to reduce the issue of following the tool vector just under the surface since the gradient forces the surface voxel search toward the surface. Additionally, the gradient helps locate a surface voxel near the haptic point so that consecutive voxels can be near each other.

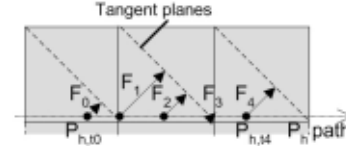


Figure 4: Force magnitude discontinuities while  $P_h$ , whose tool vector is not perpendicular to the surface slides over the surface. A frictional force artifact can also be introduced here if  $P_h$  moves from right to left.

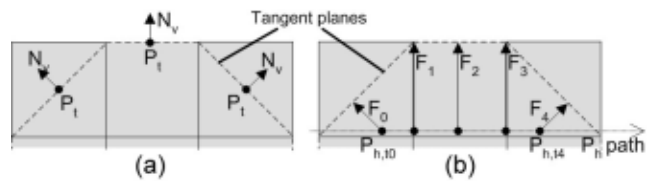


Figure 5: (a) Voxel places  $P_i$  on its boundary when exactly one facet is exposed, and at its center otherwise.  $N_v$  and  $P_i$  used for force calculations. (b) The force,  $F_i$ , for  $P_i$  as it follows the "shaved" corners.

Once a surface voxel is identified for a haptic point, a "tangent plane" is constructed to compute  $d$ . A tangent plane always runs through the center of the voxel and is perpendicular to the haptic point's tool vector with the original Voxmap point-shell method. Force magnitude discontinuities arise as voxel boundaries are crossed during sliding motion if a haptic point's tool vector is not perpendicular to the sliding plane [4, 7]. Figure 4 shows how the force of a haptic point ( $P_h$ ) changes as the point is moved across the surface. The force discontinuity can occur at every voxel because the orientation of the tangent plane and the direction of the force is based on a point's tool vector and not the shape of the surface. This often causes the haptic device to oscillate across voxel boundaries leading to instabilities in the haptic interactions. Also, a frictional force artifact is introduced in Figure 4 if  $P_h$  travels from right to left.

Since the surface gradient is known, it is possible to instead construct a plane perpendicular to the surface normal and apply a force normal to the surface. This helps reduce force discontinuities by allowing the haptic points to follow the surface shape more closely, see Figure 5b. To help construct the tangent plane, a voxel returns a point,  $P_i$ , which lies on the plane when queried.  $P_i$  is determined according to which facets of the voxel are covered or exposed to the surface. When exactly one facet is exposed, the voxel places  $P_i$  on that facet by shifting the origin of the center by  $1/2$  the voxel size in its direction. The voxel returns  $P_i$  as its center point in all other cases. Figure 5a provides a 2D example. Once  $P_i$  is found,  $d$  is computed using equation 1, where  $P_h$  is the position of the haptic point and  $N_v$  is the voxel's unit gradient vector. Since the tangent

plane can be placed within a voxel,  $d$  is set to zero if the point is above the plane. The collision force for a haptic point is then computed in the direction of  $N_v$  using equation 2. The resultant force that is sent to the haptic device is found by averaging the individually sampled forces. As shown in Figure 5, locating the tangent plane with the described method allows the plane to be placed on a voxel’s boundary when the surface is flat. Voxels on curved surfaces or corners are smoothed out to help reduce the “stair-step” feeling that will occur if the actual shape of the voxel is followed.

$$d = (P_t - P_h) \cdot N_v \quad (1)$$

$$F = N_v dk \quad (2)$$

The algorithm’s performance is mainly based on the number of haptic points used. The haptics rendering loop typically needs to run at a 1 kHz update rate in order to maintain haptic stability. Thus, the maximum number of haptic points is restricted by a 1ms constraint. Unlike small spherical bits with low numbers of haptic points, some bigger tools can require a large number of points. To support a large number of haptic points while maintaining haptic stability, we calculate collision forces at a slower rate outside the main haptic rendering loop. Significant force changes can result between slow updates, leading to mechanical instabilities if directly inserted into the haptic loop. Thus, we utilize a multi-rate haptic interface [2] to introduce new forces into the haptic loop and interpolate forces (at haptic rates) between new updates. The multi-rate haptic interfacing is based on Equation 3. The new collision force ( $f(P_N)$ ) and the local force gradient ( $\nabla f(P_N)$ ) are computed at the slower update rate based on the current haptic device position ( $P_N$ ).  $P_h$  is the device’s position at haptic update rates. A new force is interpolated within the haptic loop ( $F(P_h)$ ) using  $\nabla f(P_N)$ . The interpolated force is a first order approximation of the change in force based on the deviation of the haptic device’s position from  $P_N$ .

$$F(P_h) = f(P_N) + \nabla f(P_N)(P_h - P_N) \quad (3)$$

### 3 BONE DRILLING

Bone drilling capabilities are modeled in order to simulate the interactions between the tool bits and the bone material. A separate set of erosion points are generated for computing bone removal. Similar to the haptic point generation method described in Section 2, several parameters are used to create the erosion points based on either a sphere or cylinder. The erosion points are generated for both the surface and interior of the drill bit by voxelizing the chosen primitive shape, as shown in Figure 6. The highlighted voxels are used to generate the erosion points. Each erosion point stores its position relative to the tool bit’s center and an erosion factor that controls the amount of bone it removes. The erosion points’ positions are based on the center points of the utilized voxels. The erosion factor can be precomputed based on any erosion model. Different tool bits can be designed by varying the erosion factor value (e.g., according to its position within the bit) to define areas that can/cannot remove bone and/or remove bone at different rates. For example, the erosion factor can be based on the point’s distance from the tool’s rotational axis to consider its angular velocity. Interior points can also be set to erode bone faster than surface points so more bone is removed as a surgeon pushes harder with a tool.

Collisions between the tool bit and bone material are checked by sampling the voxmap with the erosion points. A collision is detected when a point intersects a voxel with a non-zero density value. The bone erosion model takes into account a voxel’s bone density ( $b_v$ ) and a point’s erosion factor ( $e$ ) scaled by the tool bit’s rotational speed ( $s$ ). The value of  $b_v$  is updated with equation 4 and is set to zero if it falls below a minimum threshold. A bounding box within the volume is tracked as the bone density values are modified. After processing all erosion points, the gradient values

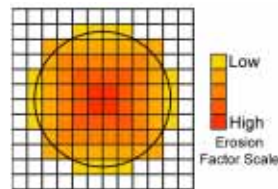


Figure 6: Voxelization of a tool bit to generate erosion points. The voxels are color coded based on its erosion factor.

for voxels within the bounding box are updated. The bone material is also visually updated as described in the following section.

$$b'_v = b_v \times (1.0 - e \times s) \quad (4)$$

### 4 VISUAL RENDERING OF BONE

The programmable graphics pipeline of current GPUs allows programs to be executed by the GPU. Fragment programs can generate color values for frame buffer. They also support texturing operations to access color values within texture maps using texture coordinates. This texture sampling capability is utilized for computing ambient and diffuse lighting to shade the volume using a fragment program. The ambient and diffuse light model used by the fragment program,  $C_s = (I_a \times C_v) + (I_d \times C_v \times \max(N \cdot L))$ , requires the ambient intensity ( $I_a$ ), diffuse intensity ( $I_d$ ), light vector ( $L$ ), surface normal ( $N$ ), and the voxel color ( $C_v$ ) to compute a shaded color value ( $C_s$ ). The values of  $I_a$ ,  $I_d$ , and  $L$  are supplied as input parameters. The volume’s surface normal is estimated using the bone density gradient. We reuse the gradients computed for haptics for shading. The RGB/RGBA components of textures typically store unsigned color values. However, textures can also store other data types encoded as colors. The volume’s gradient values are stored as a “normal map.” A normal vector in the  $[-1.0, 1.0]$  range is encoded as RGB values in the  $[0.0, 1.0]$  range with equation 5. The fragment program expands the range-compressed normal using equation 6.

$$C = (0.5 \times N) + 0.5 \quad (5)$$

$$N = 2.0 \times (C - 0.5) \quad (6)$$

An opacity map controls the transparency of a voxel and is used to visually remove bone material while drilling. The opacity value of a voxel is taken from its density value. The opacity is reduced as the density is lowered during bone removal until it becomes completely transparent. Both the opacity and gradient maps are updated when bone material is removed. To minimize the texture updates, the normal and opacity maps are combined into a single RGBA 3D texture. A bounding box is tracked during bone removal. Only the area within the bounding box is updated via a 3D sub-texture [8]. A second RGB texture specifies the bone’s color values. Multi-texturing [8] is used to specify both sets of textures and texture coordinates to the fragment program.

### 5 EXPERIMENTS

Several experiments were conducted to evaluate the performance of the described algorithms. The test system has an AMD Athlon 64 X2 4800+ dual-core processor, 2GB RAM, and two NVIDIA GeForce7800 GTX video cards (256 MB) in SLI configuration.

Haptic interactions can be computed directly within the haptic rendering loop or outside the haptic loop using the multi-rate interface. In the former case, only the force at the current haptic device position is calculated. In the latter case, both the force and force gradient are computed. Point-shells with specified numbers of haptic points are automatically generated by a test program and positioned where 1) no haptic points are intersecting a  $128^3$ -sized

voxmap (best case), and 2) where all haptic points are intersecting the voxmap (worst case). Only the initial collision detection check is performed for the best case. Collision detection, surface detection, and force calculation tasks are executed for the worst case. When computing haptic interactions within a 1 kHz haptic rendering loop, up to 729 haptic points (at 0.9ms) are supported. The remaining 0.1ms is for device communication (get position and send force). Figure 7 provides results using multi-rate haptics. Similar to [2] we were able to calculate force updates as low as 20 Hz using the multi-rate haptic interface while maintaining haptic stability. This allows up to 5,800 haptic points (695% increase) to be used. Collision detection accounts for about 9-11% of the calculation time. The remaining 89-91% of the time is required for locating surface voxels and computing force feedback.

The maximum number of erosion points that can be used for a tool bit can be estimated based on the time required to compute bone removal, update the gradient information, update the normal/opacity 3D texture for visual updates, and the desired update rate. Similar to the haptic rendering experiments, erosion points are automatically generated by the test program. Only the worst case is considered since it includes both collision detection and updates to the bone density. Bone erosion is computed in constant time (0.142  $\mu$ s) per erosion point. The size of the bounding box modified while drilling dictates the time required for recomputing gradients and updating the 3D bone texture. Figure 8 provides results for various bounding box sizes. The maximum number of erosion points for a bit can be estimated using Figure 8 and the bone erosion time. For example, up to 221,700 erosion points can be used for a tool bit that can modify a maximum bounding box size of  $20^3$  voxels with a 30 Hz update rate requirement.

The final set of experiments help determine the graphics update rates with different volume sizes and the cost of computing volumetric shading. Several  $2^N$  volume sizes (texture size requirement) were tested. The volumes were rendered using object-aligned slices [3]. A volume up to  $256^3$  voxels can be rendered in real-time (at 41Hz). A  $128^3$  rendered at 304Hz and 308Hz with and without shading, respectively. Shading dropped the rendering rates by less than 1.5% for all tested volume sizes.

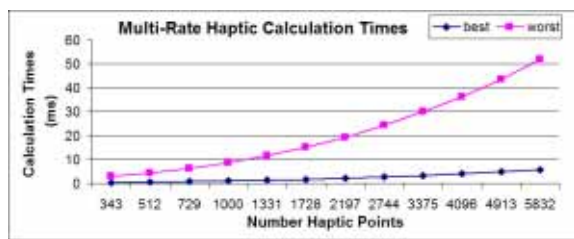


Figure 7: Haptic interaction results with multi-rate haptics.

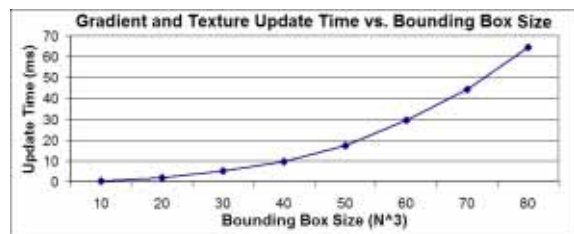


Figure 8: Results for updating gradients and 3D sub-texture.

## 6 CONCLUSION/DISCUSSION

This paper describes the haptic and visual algorithms created to simulate bone cutting tools for voxel-based bone material. The improved stability of the Voxmap point-shell haptic algorithm can be felt with the haptic device during haptic interactions. Additional testing will allow us to quantify the improvements. The maximum number of haptic points supported by a tool bit (while maintaining haptic stability) will vary with different computer systems. The 1ms haptic loop constraint only allows small force calculation time fluctuations. The multi-rate haptic interface helps maintain stable haptic interactions across different systems without adjusting tool bits' point-shells. Using texture-based volume rendering to display the bone in real-time, and performing shading calculations on the GPU helps free up the CPU for other simulator tasks.

The described algorithms have been tested for drilling on a voxelized block. It will be possible to generate the voxmap directly from the voxel discretization of patient-specific 3D CT and MR datasets to generate virtual patients for a surgical simulator.

Several surgical tools that are typically used during a craniotomy have been simulated. Separation of the haptic and erosion points enables modeling the different types of tool bits. Realistic 3D models are created from real surgical tools and controlled by the haptic device during their use. To create realistic tools, we can try tuning the erosion factors and haptic response based on experimental data and/or expert feedback. The generality of the described algorithms will make it possible to create additional virtual tools.

## ACKNOWLEDGEMENTS

We would like to thank Penny Christian from Medtronic and Jason Martin from Stryker for lending us the sets of surgical tools used to model the virtual instruments for this work.

This work is supported by the U.S. Army Medical Research and Materiel Command under Contract No. W81WH-05-C-0142. The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision unless so designated by other documentation.

## REFERENCES

- [1] M. Agus, A. Giachetti, E. Gobbetti, G. Zanetti, and A. Zorcolo. Real-time haptic and visual simulation of bone dissection. *Presence*, 12(1):110–122, 2003.
- [2] M. C. Cavusoglu and F. Tendick. Multirate simulation for high fidelity haptic interaction with deformable objects in virtual environments. In *IEEE International Conference on Robotics and Automation (ICRA 2000)*, pages 2458–2465, 2000.
- [3] K. Engel, M. Hadwiger, J. Kniss, and et al. High-quality volume graphics on consumer pc hardware. In *SIGGRAPH Course Notes*, 2002.
- [4] W. McNeely, K. D. Puterbaugh, and J. J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *ACM SIGGRAPH*, pages 401–408, 1999.
- [5] D. Morris, C. Sewell, N. Blevins, F. Barbagli, and K. Salisbury. A collaborative virtual environment for the simulation of temporal bone surgery. In *MICCAI*, 2004.
- [6] A. Petersik, B. Pflesser, U. Tiede, K. H. Hohne, and R. Leuwer. Haptic volume interaction with anatomic models at sub-voxel resolution. In *IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 66–72, 2002.
- [7] M. Renz, C. Preusche, M. Potke, H. Kriegel, and G. Hirzinger. Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. In *Eurohaptics*, pages 149–154, 2001.
- [8] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, Fourth Edition*. Addison-Wesley, 2004.
- [9] D. Stevenson, K. Smith, J. McLaughlin, C. Gunn, and et al. Haptic workbench: A multisensory virtual environment. In *SPIE Stereoscopic Displays and Virtual Reality Systems VI*, pages 356–366, 1999.