Burrhole Simulation for an Intracranial Hematoma Simulator

Eric ACOSTA [a,1], Alan LIU [a], Rocco ARMONDA [b], Mike FIORILL [c], Randy HALUCK [c], Carol LAKE [c], Gilbert MUNIZ [a], and Mark BOWYER [a]

[a] *The National Capital Area Medical Simulation Center, Uniformed Services University*
[b] *National Capital Neurosurgery Consortium, National Naval Medical Center*
[c] *Verefi Technologies, Inc.*

Abstract. Traumatic head injuries can cause internal bleeding within the brain. The resulting hematoma can elevate intracranial pressure, leading to complications and death if left untreated. A craniotomy may be required when conservative measures are ineffective. To augment conventional surgical training, a Virtual Reality-based intracranial hematoma simulator is being developed. A critical step in performing a craniotomy involves cutting burrholes in the skull. This paper describes volumetric-based haptic and visual algorithms developed to simulate burrhole creation for the simulator. The described algorithms make it possible to simulate several surgical tools typically used for a craniotomy.

Keywords. Surgical simulation, bone drilling, volume rendering, haptic feedback

Introduction

Head trauma commonly occurs on the battlefield. Resulting brain injuries and bleeding can elevate intracranial pressure, leading to complications and death if not treated. Training for head injury treatment is difficult to come by under battlefield conditions. Neurosurgery is a specialized skill that requires extensive training. Current training occurs on live patients. A surgical simulator can augment current training methods so trainees become proficient at the required surgical skills before working on the first patient. A craniotomy may be required for treatment when conservative measures are ineffective. This procedure involves removing a section of the skull in order to gain access to the brain. We are developing a Virtual Reality-based training simulator to practice this skill. A haptic workbench [9] is used to generate a virtual environment with 3D stereoscopic visual and haptic feedback. Surgical tools are controlled with a haptic device. The workbench allows the visual and haptic workspaces to be co-registered to preserve hand-eye coordination for surgical training. An important step of a craniotomy involves cutting burrholes in the skull using powered tools. This paper describes volumetric haptic, bone erosion, and visual algorithms developed to simulate bone cutting tools for the simulator.

The remainder of the paper is as follows. Section 1 describes the haptic rendering algorithm used in the simulator. Section 2 provides details of our method for computing

[1]Correspondence to: Eric Acosta, http://simcen.usuhs.mil; E-mail: eacosta@simcen.usuhs.mil.

bone erosion. Section 3 documents the visual rendering algorithms. Section 4 shows several surgical tools simulated with the described algorithms and the paper is concluded in Section 5.

1. Haptic rendering

A modified Voxmap point-shell algorithm [5,7] is created to simulate haptic interactions between bone cutting tools and bone. The original method used to compute force feedback in [5] and [7] can introduce considerable force discontinuities at voxel boundaries. We address the stability of the Voxmap point-shell haptic algorithm by modifying its surface boundary detection and force feedback calculation methods.

The haptic rendering algorithm represents a virtual environment using a spatial occupancy map called a voxmap. Bone is encoded within the voxmap using voxels. Each voxel encodes the bone density, density gradient vector, and color at its location. The tool bits are modeled as a set of haptic points that are spatially distributed to define the bits' shapes and sizes. The haptic points form a "point-shell" that approximates the surface of a tool bit. Each haptic point stores its position relative to the tool bit's center and an inward pointing tool normal that is used for calculating a collision force. The spatial positioning of the haptic points impacts force calculation [7]. To ensure even and symmetric point distributions, the positions and tool normals of haptic points are computed based on either a spherical or a cylindrical approximation of a tool bit. Several parameters, such as tool radius, height, and voxel size etc., help generate point-shells for bits of different shapes and sizes. Figure 1a shows the point-shell of a perforator and a round bit. The number of haptic points created for a tool bit is also controlled using the parameters.

Collisions between the haptic device and the virtual environment are checked by probing the voxmap with the haptic points. A collision occurs when a haptic point in-

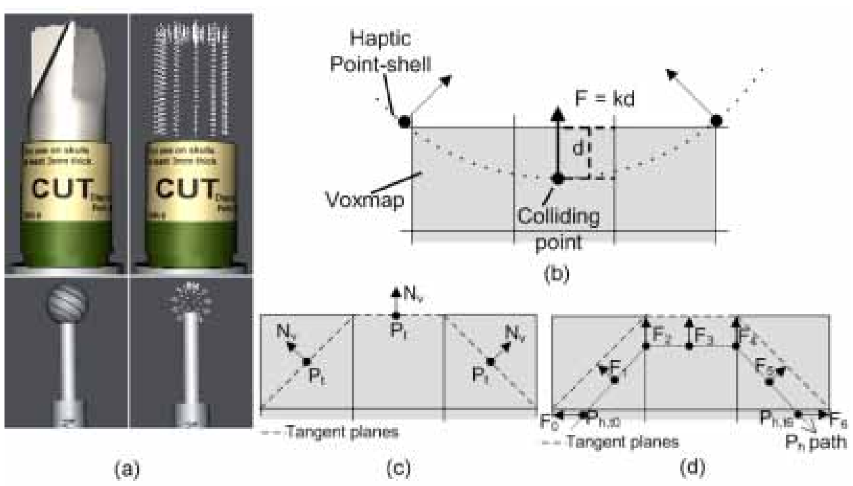

Figure 1. (a) Haptic point-shell approximations for a perforator bit (316 haptic points) and round bit (72 haptic points). (b) Point-shell interaction with a voxmap. (c) N_v and P_t used to construct "tangent planes" for force calculations. P_t placed on voxel boundary when exactly one facet exposed to surface, and at voxel center otherwise. (d) The force, F, for a haptic point, P_h, as it follows the bone's surface.

tersects a voxel with a density value greater than zero. A force is calculated for each colliding haptic point based on Hooke's law $F = kd$, where k is a stiffness constant and d is the point's penetration depth within the bone's surface. Figure 1b illustrates.

To help locate the surface for a haptic point, its tool normal is followed at voxel-sized intervals until a voxel with a non-zero gradient is encountered. The surface is then located by sampling along the gradient direction. Since bone material is stiff, little penetration is expected, and only a few samples are normally required. To avoid rendering artifacts, a threshold on the number of samples (e.g. 10 samples) is imposed. A haptic point is ignored if the bone's surface cannot be found within the set number of samples. In [5,7], voxels are labeled as interior, surface, or free. Since bone material is removed while drilling, a surface detection algorithm would be required in our case to dynamically update voxels' status. Instead, the currently sampled non-empty voxel is considered to be a surface voxel when the next voxel along the sampling vector has a zero density value.

Once a surface voxel is located, a "tangent plane" is constructed to compute d. With the original Voxmap point-shell algorithm, a force discontinuity can occur at every voxel for a haptic point whose tool normal is not perpendicular to the surface [5,7]. The discontinuities occur because the tangent plane's orientation and the computed force is based on the tool normal direction and not the surface's shape. We use the surface gradient to instead construct a plane perpendicular to the surface normal and to apply a force normal to the surface. A voxel returns a point, P_t, which lies on the tangent plane. P_t is placed on the voxel's surface boundary when exactly one facet is exposed. P_t is a voxel's center point in all other cases. Figure 1c is an example. Equation 1 is used to compute d as the distance from the haptic point (P_h) to the tangent plane. N_v is a voxel's unit gradient vector. The value of d is set to zero if the haptic point is above the plane. A haptic point's force is then computed in the direction of N_v with equation 2. The resultant force for the haptic device is the average of the colliding haptic points' forces.

$$d = (P_t - P_h) \cdot N_v \tag{1}$$

$$F = N_v k d \tag{2}$$

As shown in Figure 1d, force discontinuities are reduced by allowing a haptic point to transition across voxel boundaries. Additionally, voxels on curved surfaces or corners are smoothed out to reduce the "stair-step" feeling that is typical of voxel-based haptic interactions.

2. Bone drilling

Many existing bone drilling methods, such as [2] and [6], combine the voxel sampling methods required for haptics and bone erosion. This works well for spherical drill bits since all areas can cut bone. To model different types of tool bits, we separate the haptic and bone erosion sampling points.

A set of erosion points are generated to simulate the bone drilling capabilities of tools. Haptic points are only required along a tool bit's surface to compute a force response, as described in Section 1. However, it is necessary to generate interior and surface erosion points to prevent leaving residual bone material behind when the tool bit penetrates the bone's surface. Erosion points are generated by voxelizing either a sphere or cylinder. An erosion point is generated for each voxel whose center point falls within the boundaries of the chosen primitive shape, as shown in Figure 2. Parameters, such as tool radius, height, and voxel size etc., help control the size and shape of the tool bit. Erosion points store their position relative to the tool bit's center and an erosion factor that determines the amount of bone it can remove based on the rotational speed of the bit. An erosion point's position is based on the center point of its corresponding voxel during primitive voxelization. The erosion factor can be precomputed based on any erosion model. Different tool bits can be modeled by varying the erosion factor value, based on its position within a bit, to define areas that can/cannot remove bone and/or remove bone at different rates. For example, a perforator's cutting surface is restricted to its blades, whereas a ball bit has a spherical cutting surface. Interior points also erode bone faster than surface points to remove bone quicker as a surgeon applies more force with a tool.

A collision between the tool bit and bone material is detected when an erosion point intersects a voxel with a non-zero density value. The bone density at the colliding voxel (B_v) is reduced according to equation 3, where e is the point's erosion factor and s is the tool bit's rotational speed. The value of B_v is set to zero once it falls below a minimum threshold. A bounding box within the voxmap is tracked as bone density values are modified. The gradient values for the voxels within the bounding box are updated once all erosion points are processed. The bone material is also visually updated.

$$B'_v = B_v(1.0 - e \times s) \tag{3}$$

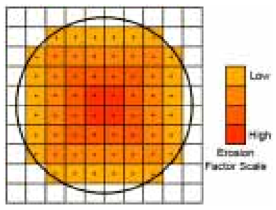

Figure 2. Erosion points generated by voxelizing tool bit shape. Voxels are color coded according to erosion factor.

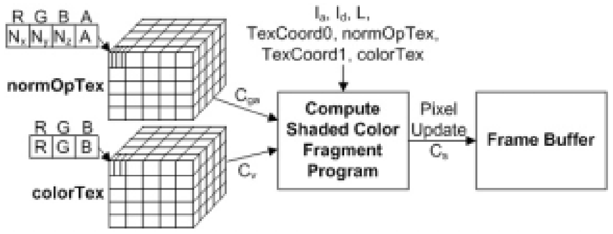

Figure 3. RGBA normal/opacity map, RGB color texture, texture coordinates, and lighting parameters used by GPU fragment program to generate shaded color values.

3. Visual rendering

Three-dimensional texture-based volume rendering [3] is used for real-time visual display of bone. A fragment program [4] is created to compute volumetric shading on the video card's GPU as the volume is rendered. The program generates shaded color values that become pixels in the rendered frame buffer image, Figure 3. A fragment program can perform mathematical operations on the values stored within texture maps. Texture coordinates are used to access the color values within textures. This texture sampling capability makes it possible to compute the ambient and diffuse lighting to shade the volume. The following algorithm is implemented in the fragment program to shade voxels as they are rendered:

1. Get voxel color (C_v) by sampling colorTex.
2. Get voxel gradient and opacity (C_{ga}) from normOpTex and assign opacity to output color's (C_s) alpha component.
3. Compute shaded RGB components for C_s using the light model: $(I_a \times C_v) + (I_d \times C_v \times max(N \cdot L))$, where I_a is the ambient intensity, I_d the diffuse intensity, L the light vector, and N the surface normal acquired by expanding the gradient vector of C_{ga} with Equation 5.

The surface normal is estimated using the bone density gradient. The volume's gradient is stored in a special texture called normal map. Equation 4 is used to range compress normalized gradient vectors from a [-1.0, 1.0] range to an unsigned color value range of [0.0, 1.0] in order to encode the vectors into the normal map. The fragment program expands the range-compressed normals with Equation 5.

$$C = (0.5 \times N) + 0.5 \tag{4}$$

$$N = 2.0 \times (C - 0.5) \tag{5}$$

An opacity map is used to control the visibility of voxels. The voxels' opacity values are taken from the bone density values. Voxel opacity values are reduced while drilling until they become transparent due to bone erosion. The opacity and normal maps are updated when the volume is modified while drilling. To minimize texture updates, both maps are combined into a single RGBA three-dimensional texture, Figure 3. A sub-texture is used to only update values that fall within a modified bounding box region, which is tracked by the bone drilling algorithm. Color information for the bone is specified using a second RGB texture. The two textures and their texture coordinates are simultaneously specified to the fragment program using multi-texturing [8]. The lighting values are specified to the fragment program as input parameters.

4. Virtual tool simulation

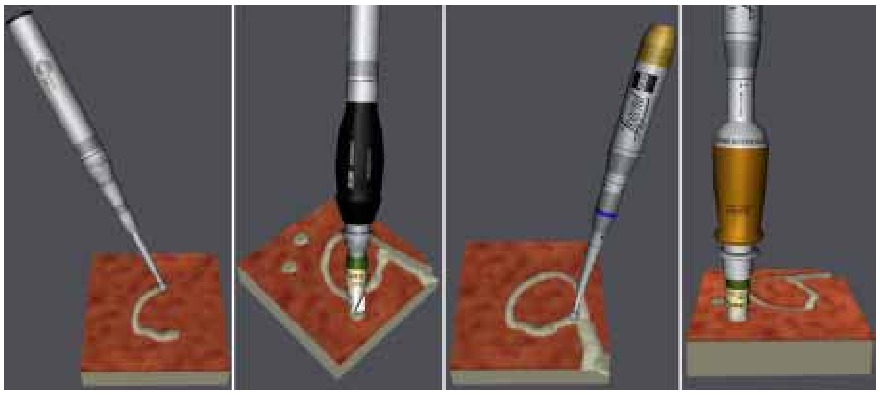

Figure 4. 3D models of tools typically used in clinical practice. (Left) Bone drill and perforator from Stryker. (Right) Bone drill and perforator from Medtronic.

The described algorithms make it possible to simulate several surgical tools, such as bone drills and perforators, typically used for a craniotomy. A haptic workbench [9] is used to generate a virtual environment with 3D stereoscopic visual and haptic feedback. Figure 4 shows realistic 3D models that are created from real surgical tools and controlled by a PHANTOM haptic device during their use.

5. Conclusion

A simulator can be a vital tool to help train for the difficulties and complications surrounding intracranial hematoma surgery. We have taken the first steps towards creating a simulator by developing the algorithms needed to simulate virtual tools for making burrholes. The generality of the methods used to model the tool bits and bit-bone interactions will make it possible for additional virtual tools to be simulated. The algorithms have been demonstrated for drilling on voxelized blocks. However, it will be possible to generate the voxmap directly from the voxel discretization of 3D CT and MR datasets to generate different virtual patients for a surgical simulator. Development of other surgical effects is also underway for the simulator.

Acknowledgments

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 tools.

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] R.J. Adams and B. Hannaford. Stable haptic interaction with virtual environments. *IEEE Transactions on Robotics and Automation*, **15**(3), (1999), 465–474.

[2] M. Agus, A. Giachetti, E. Gobbetti, G. Zanetti, and A. Zorcolo. Real-time haptic and visual simulation of bone dissection. *Presence*, **12**(1), (2003), 110–122.

[3] K. Engel and M. Hadwiger and J. Kniss and et al. High-quality volume graphics on consumer PC hardware. *SIGGRAPH Course Notes*, (2002).

[4] R. Fernando and M.J. Kilgard. *The Cg Tutorial*, Addison-Wesley, 2003.

[5] W.A. McNeely, K. D. Puterbaugh, and J. J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. *ACM SIGGRAPH*, (1999), 401–408.

[6] D. Morris, C. Sewell, N. Blevins, F. Barbagli, and K. Salisbury. A collaborative virtual environment for the simulation of temporal bone surgery. *Medical Image Computing and Computer-Aided Intervention*, (2004).

[7] M. Renz, C. Preusche, M. Potke, H.P. Kriegel, and G. Hirzinger. Stable Haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. *Eurohaptics*, (2001), 149–154.

[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. *PIE Stereoscopic Displays and Virtual Reality Systems VI*, **3639**, (1999), 356–366.