Real-time interactions and synchronization of voxel-based collaborative virtual environments

Eric Acosta*, Alan Liu

National Capital Area Medical Simulation Center
Uniformed Services University

ABSTRACT

Collaborative virtual environments (C-VE) facilitate team-oriented training on Virtual Reality-based surgical simulators. Many C-VEs replicate the VE on each user's machine to allow for real-time interactions. However, this solution does not work well when modifying voxel-based C-VEs because large and frequent volumetric updates make it difficult to synchronize the C-VE. This paper describes a hybrid depth-buffered image (DBI) and geometry-based rendering method created to simulate visual interactions between local virtual bone cutting tools and remotely maintained volumetric bone material for a craniotomy simulator. For real-time interactions, users only store a DBI of the volumetric C-VE and composite it with rendered images of surgical tools. Additionally, we describe methods to combat network bandwidth/latency to remotely simulate haptic and bone drilling interactions between users' tools and the volumetric VE. For haptic feedback, a multi-rate solution [9] allows users to construct a local approximation of the volumetric C-VE to compute new forces. Only 2D DBI updates are required to synchronize different users when the bone changes due to drilling. Our approach provides an improved performance over a replicated VE that uses 3D model-based updates.

Keywords: Collaborative virtual environment, virtual reality, depth buffered image, surgical simulation, remote volume rendering.

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

Virtual Reality-based medical surgical simulators enable surgeons to train on virtual patients. Many existing surgical simulators are stand-alone systems and focus on training a single user at a time. Collaborative Virtual Environments (C-VEs) enable multiple participants, which are often remotely located, to interact within a shared virtual world. This capability opens up a wide array of possibilities in many areas such as the medical field. Several efforts have been directed toward leveraging the capabilities of C-VEs for collaborative training [17, 14, 18, 13]. Our motivation for the described work is the development of a Virtual Reality-based surgical simulator to assist with training for a craniotomy procedure. A craniotomy is a surgical procedure to remove a section of the skull in order to access the brain for treatment. In [4], volumetric haptic, bone erosion, and visual algorithms have been developed to simulate the interactions between virtual bone cutting tools and voxel-based bone material for a stand-alone system. The algorithms make it possible to simulate several surgical tools, such as the bone drill and perforator shown in Figure 1, for a craniotomy.

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

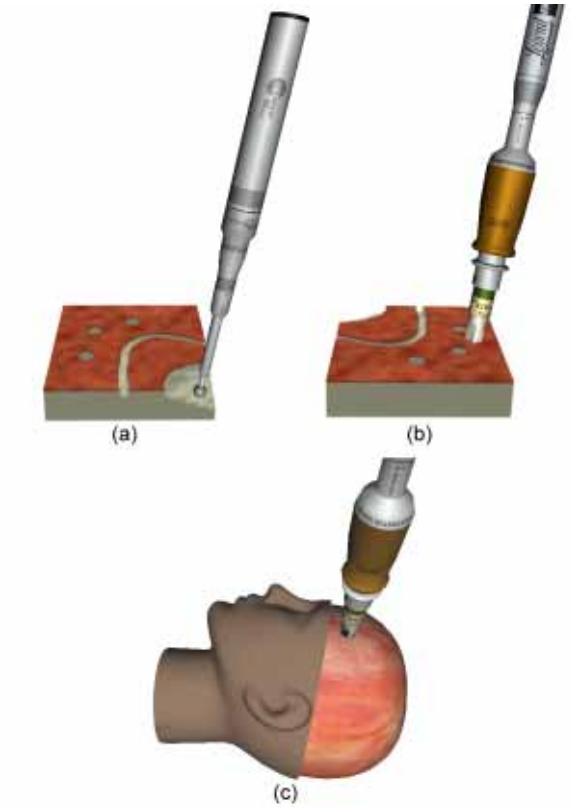

Figure 1: Example simulated (a) bone drill and (b) perforator surgical tools drilling into voxelized block representing bone material. (c) Perforator cutting burrhole in virtual patient's skull.

This paper extends our previous work to support networked interactions for collaborative surgical training. Training multiple users typically requires the use of multiple computer systems, which are connected through a local or wide-area network. The separation of the users (and the connection medium joining the different systems) inevitably introduces delays within shared interactions. Many C-VEs maximize local responsiveness of the interactions through localization and database replication [20, 16]. Maintaining local copies of the C-VE on different systems requires a consistency control method to synchronize each user's local snapshot of the VE. In C-VEs, users' reactions and decisions are based on the current view of the shared environment. Thus, it is critical for medical C-VEs to use a fairly strong consistency model to minimize discrepancies in each VE.

Recently, Hutchins et al. [14] and Morris et al. [18] described collaborative bone drilling capabilities between two users for temporal bone surgery. Both of their approaches replicate the volumetric VE on each user's system and send volumetric model updates

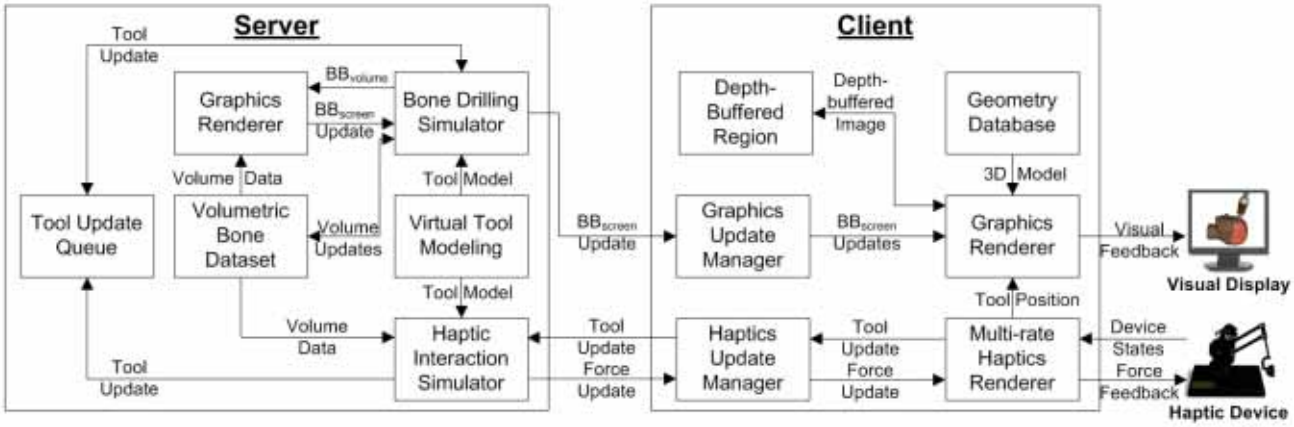

Figure 2: Collaborative virtual environment client-server architecture.

over a local Ethernet network to synchronize the volumetic C-VE. We found that maintaining local copies of the volumetric VE on each user's system makes it difficult to synchronize the C-VE. The volumetric VE is modified at the voxel-level as the user drills away bone material. Modifications to the bone adds significant processing overhead for users' systems to handle large frequent updates from other users. Additionally, some lower-end systems may lack the computational and/or rendering capacity to handle the volumetric VE. Our synchronization solution is to centralize the volumetric VE within a central server. Maintaining the volumetric VE on a single machine allows us to avoid using consistency control mechanisms such as ownership locking and transfer or sequential sharing [20]. The major drawback to centralizing the volumetric VE is that remote rendering and interactions is limited by network latency. This makes it difficult to interact with the VE in real-time. To overcome this issue, we use a hybrid depth-buffered image (DBI) and geometry-based rendering method in order to combine remote volume rendering of the skull with local geometry rendering of virtual tools and patient anatomy. A method is presented to correctly create DBIs with texture-based volume rendering. The described work makes it possible for the tools to visually interact with the bone material in real-time. Furthermore, since the server maintains the only copy of the volumetric VE, we describe methods to combat network bandwidth and latency issues to remotely simulate haptic and bone drilling interactions between clients' tools and the volumetric VE in real-time. Only two-dimensional visual updates (of the modified areas within rendered volume images) are required to synchronize clients' local DBI representation of the volumetric C-VE.

The remainder of the paper is organized as follows. Section 2 overviews related work. Our volumetric C-VE approach is described in Section 3 and experimental results are given in Section 4. The paper is then concluded in Section 5.

2 Related Work

Considerable work has been performed for simulating bone drilling on stand-alone systems. Aguas et al [6, 5] developed a physically-based visual and haptic model for bone drilling. Petersik et al. [19] developed a method to simulate bone drilling interactions at the sub-voxel level. Most existing work has focused on simulating spherical drill bits for temporal bone surgery. In [4], we describe haptic and visual algorithms to easily simulate different types of bone drilling tools used to perform a craniotomy. Additional bone drilling simulation works can be found in the literature.

Limited work has also been performed for collaborative bone drilling environments. Morris et al. [18] and Hutchins et al. [14] allow two users to connect together over a network for collaborative interactions. The C-VE is useful to allow an instructor to monitor a trainee's performance and interactively provide feedback. In [18], a user can also choose to drive their local haptic device using the forces generated by the remote participant's drill. [14] provides additional tools, such as a marker and eraser, to allow the instructor or student to draw on the scene. Both of their setups replicate the volumetric temporal bone VE locally and require the bone model to be updated at the volumetric-level for synchronization. Additionally, a second local polygonal mesh-based representation needs to be updated for visual display. This 3D model-based approach can be very computationally expensive, especially as the modified volume size increases. Additionally, only systems that provide adequate computational and rendering capabilities to handle the entire C-VE can be used to run the simulators.

In addition to these collaborative bone drilling environments, work has been performed to support collaborative haptic interactions. Montgomery et al. [17] have developed a framework for surgical simulation development. The framework provides a method to remotely communicate with haptic devices and can be used to develop C-VEs. A mechanism is also provided for capturing the screen image, compressing it, and sending it to remote users for display replication. Kim et al [15] were able to achieve haptic collaboration across the Atlantic Ocean for a simple virtual box lifting task. A predictive collision detection algorithm and three layers of damping (in the box's dynamics, between the virtual cube and the haptic device, and to remove hand tremor) made the task possible. Gunn et al. [13] have developed a surgical training prototype for a cholecystectomy, which allows an instructor and remote student to work in the same VE. Their work utilizes a "pseudo-physics" model to help control instabilities over long distance haptic collaboration. The model is able to withstand large latencies for highly damped soft objects. The work was demonstrated for haptic collaborations between Australia and California. Unfortunately, these works depend on having a local copy of the VE available. In our case, the users do not have a copy of the volumetric bone, so a multi-rate haptic interface method [9] that was originally developed for deformable objects is utilized to approximate the volumetric model to compute haptic feedback. Additional details are given in Section 3.

Previous work on remote volume rendering and image-based rendering exists. The framework described by Engel et al. [11] and SGI's VizServer [3] maintains and renders a volumetric VE on a central server. User interface events from clients are transmitted to the server and resulting images are returned for display. With this setup, clients can leverage the capabilities of more powerful servers. However, low responsiveness in the interactions may occur due to network delays. This performance trade-off is unacceptable

for our application. Our surgical tools need to be able to interact with the volumetric VE in real-time.

Image-based rendering provides an alternative to traditional geometry-based rendering methods. Many image-based rendering techniques exist [22, 23]. One major advantage of image-based rendering techniques over geometry-based rendering is that the image can be rendered independent of the VE complexity. Instead, rendering performance is dependent on the image resolution used. This makes real-time interactions with complex scenes possible. For example, for an architectural walk through [8] and a city scene [24] an image can replace distant parts of the scene. Geometrical models can then replace the images when the user approaches the geometry depicted in the images. Depth-buffered images (DBI) extend RGB color images to include a depth component. DBIs are commonly used to distribute large datasets and complex scenes over different rendering nodes and then composite partial images into final images [7, 25, 26].

3 Collaborative virtual environment

In our application, we made the observations that most of the volume remains unchanged from frame-to-frame, and modifications only occur to areas where a tool cuts away bone material. On the other hand, the position and orientation of the virtual tools can change every frame. Also, the viewpoint of the VE remains relatively fixed and does not change often. Based on these observations, we are able to maintain a volumetric VE representing the skull remotely and the virtual tools and other patient anatomy locally. Users only maintain a DBI of the volumetric bone VE. The DBI is remotely generated by a server that manages the volumetric VE. The DBI is then composited locally with rendered surgical tools and patient anatomy.

3.1 Architectural overview

The client-server architecture in Figure 2 is used to establish the C-VE. The server is dedicated to maintaining the volumetric VE and performing the calculations necessary to simulate virtual tool interactions. A user's system is considered a client.

The server consists of six main modules. The *Volumetric Bone Dataset* is the voxel-based dataset used to model the bone material. The representation of the virtual tools [4], which is used to compute haptic interactions and bone drilling, is maintained within the *Virtual Tool Modeling*. The *Haptic Interaction Simulator* receives tool update data, such as position/orientation and drilling speed, to compute haptic interactions between a virtual tool and the bone material. A force is generated based on the haptic interaction and sent to the client. The tool update data are added to a *Tool Update Queue* for the server to compute bone drilling. The *Bone Drilling Simulator* checks the queue for new tool updates and computes bone drilling between the virtual tool and the bone material. If the bone is modified while drilling, the *Graphics Renderer* generates a new DBI from the volume and updates the clients.

A client's system also contains six main modules. The *Depth-Buffered Region* maintains a DBI (for the entire window) of the volumetric VE. The *Graphics Update Manager* receives DBI updates from the server and applies them to the *Depth-Buffered Region*. Realistic 3D models of bone cutting tools have been created from real surgical tools and are stored within the *Geometry Database*. The database also stores 3D models of virtual patient anatomy. A *Graphics Renderer* renders the 3D models of the user's current tool and the patient, and composites them with the *Depth-Buffered Region* to generate visual feedback. A PHANTOM haptic device [2] provides haptic feedback to the user based on interactions between a virtual tool and the bone material. The *Haptics Update Manager* sends tool updates to the server's *Haptic Interaction Simulator* and receives new force updates. The force updates are utilized by the *Multi-rate Haptics Renderer* in order to interpolate new forces between force updates from the server.

```
computeDepthValue( P_TC, ModelViewProjMatrix,
                   VolumeDim )

  if voxel < transparent threshold then
     V_Depth = 1.0
  else
     //Texture coordinates to object coordinates
     P_OC.x = (P_TC.x - 0.5) * VolumeDim.x
     P_OC.y = (0.5 - P_TC.y) * VolumeDim.y
     P_OC.z = (0.5 - P_TC.z) * VolumeDim.z
     P_OC.w = 1.0

     //Apply model view and projection matrices
     P_PC = ModelViewProjMatrix * P_OC

     //Perspective division and map into [0,1] range
     V_Depth = ((P_PC.z / P_PC.w) * 0.5) + 0.5
  end if

  return V_Depth
```

Figure 3: Fragment program to correct depth values.

3.2 Volumetric bone environment

The volumetric bone VE is maintained on the server. The client does not store a local copy of the volumetric VE. Instead, a DBI is used to visually represent the volumetric VE on the client's system. The DBI stores the RGB color and z-depth information for each pixel. The server's *Graphics Renderer* generates new DBIs for the bone in order to update clients.

Three-dimensional texture-based volume rendering is used to generate shaded volumetric images of the bone [4]. To utilize a video card's texturing hardware for volume rendering, datasets are represented as texture maps and drawn as a stack of textured polygon slices [10]. As the textured slices are rendered back-to-front, they are blended together by the graphics hardware to generate an image. The z-buffer stores the distance from the virtual camera to each rendered pixel in the frame buffer. The z-buffer values are used for hidden surface removal so that objects closer to the user's view obstruct the objects behind it. For texture-based volume rendering, the GPU generates the depth values for the z-buffer based on the coordinates of the polygonal slices. The same depth calculations are performed for all slices (and all portions of the slices) regardless if they are transparent or opaque. This presents a problem when trying to composite the DBI with a client's local geometrical 3D models (e.g., virtual tools). Transparent voxels can incorrectly obstruct a virtual tool if their corresponding polygonal slices are closer to the viewer than the virtual tool. This issue is easily resolved on a stand-alone system by rendering the tool before the volume. Although, the depth-buffer values are still incorrect, the alpha blending process creates correct images. Unfortunately, this

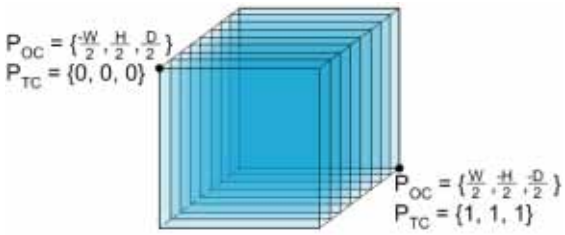

Figure 4: Volume's object (P_{OC}) and 3D texture (P_{TC}) coordinates.

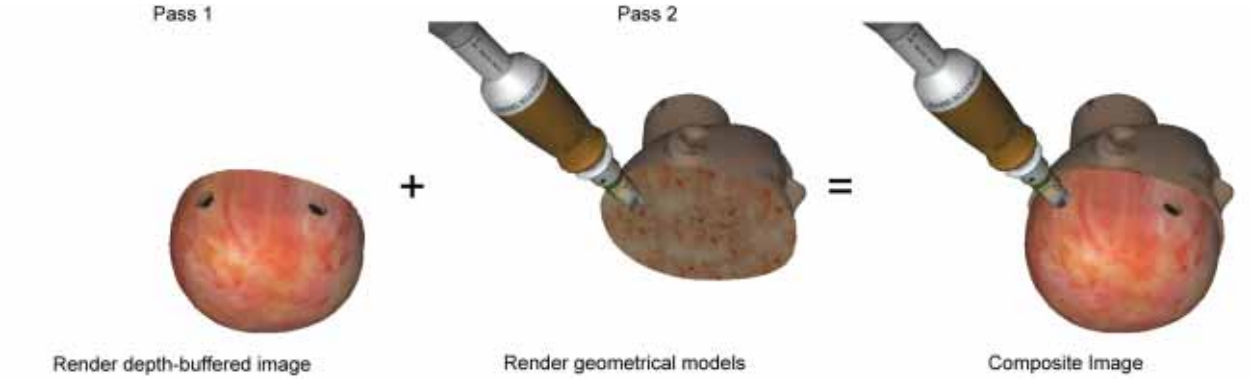

Figure 5: Two-pass rendering algorithm to display the C-VE on the client. Pass 1 restores the DBI of the volumetric VE. Pass 2 renders the surgical tool and the remainder of the patient's head. Finally the two images are composited by the graphics hardware.

approach is not possible with our setup since the client does not keep a local copy of the volume. The graphics rendering pipeline needs to be modified to generate correct DBIs of the volume.

The programmable graphics pipeline of modern GPUs allow Cg programs [12] to be written and executed by the GPU. A fragment program is used in [4] to access the volume's color, opacity, and surface normal information to shade the volume as it is rendered. Fragment programs can also overwrite the depth values of rendered geometry for the z-buffer. We utilize this capability in order to correctly generate DBIs for texture-based volume rendering. The method works for both object-aligned and view-aligned slices. The pseudo-code for the fragment program is given in Figure 3.

As the textured slices are rendered, texture coordinates of voxels are passed to the fragment program. If a voxel is transparent then the depth value is simply set to the largest possible depth value of 1.0. Otherwise, the fragment program uses the texture coordinates to compute the depth value. Figure 4 shows the object coordinates (P_{OC}) and 3D texture coordinates (P_{TC}) of the two opposite corner points. Texture coordinates are in the range of [0.0, 1.0]. The object is centered about its local origin, and the slice coordinates are based on the volume's dimensions. To compute the depth value, P_{TC} is first mapped into a [-0.5, 0.5] range. This new value is then multiplied with the volume's dimensions to get P_{OC}. As shown in Figure 3, the x, and y/z coordinates of P_{TC} are treated as different cases to correctly map from the texture coordinate system to the object's coordinate system. P_{OC} is then applied the current Modelview and Projection matrices, which are acquired from the OpenGL API, to compute the screen projected coordinate (P_{PC}). Finally, P_{PC} is converted into normalized device coordinates by performing a perspective division and the z depth value is mapped into the [0.0, 1.0] range.

The z-buffer will contain the correct depth values once the volume is rendered. The RGB color values reside in a color buffer. The (R,G,B,z) values for the DBI are acquired from both buffers using the *glReadPixels* OpenGL function.

3.3 Hybrid image/geometry-based rendering

A two-pass image/geometry hybrid rendering method is used to display the C-VE on the client's system, Figure 5. The first pass renders the DBI. The OpenGL *glDrawPixels* function is used to transfer the image from main memory to the frame buffer. When the server creates a DBI, it utilizes the same perspective projection as the client. Images rendered with *glDrawPixels* undergo similar transformations as when rendering geometry. Thus, to avoid incorrectly adding perspective warping to the image, an orthogonal projection is used when rendering the image. The original perspective projection is restored once the image is rendered. The *glDrawPixels* image rendering method is very expensive since a large amount of pixel data needs to be transferred over the system bus. To overcome this rendering bottleneck, an OpenGL buffer region is used to save the DBI into off-screen memory [1]. The buffered region allows the DBI to be restored into the frame buffer at very high frame rates. As described in Section 3.4.2, *glDrawPixels* is only used when updating the DBI. The second pass renders geometrical models of the virtual surgical tools and virtual patient. The graphics hardware composites the rendered geometry with the DBI that is already in the frame buffer to create the final image.

3.4 Networked virtual tools simulation

Since the server maintains the only copy of the volumetric VE, it computes the interactions between clients' virtual tools and the bone material. Thus, we need to utilize methods to overcome network bandwidth and latency issues so users can interact with the volumetric VE in real-time.

3.4.1 Haptic interactions

The haptics rendering loop typically requires a 1 kHz update rate in order to maintain haptic stability. Thus, the force calculation time is restricted by a 1ms time constraint. It is difficult to meet this requirement since the haptic device is controlled by a client, while the volumetric VE is on the server. Two network communications are required to send tool information from the client to the server, and then send computed forces from the server to the client. To maintain haptic stability, the force feedback is calculated at a slower rate outside the main haptic rendering loop. A client's *Haptics Update Manager* sends a tool's position/orientation and drilling speed to the server. The server's *Haptic Interaction Simulator* utilizes this information to compute new collision forces. The server maintains a local representation of the virtual tools [4] to compute haptic interactions. Computing interactions with the rigid bone material at slower rates can lead to significant force changes between consecutive updates. The new forces cannot be directly inserted into the haptic loop on the client's system because mechanical instabilities can be introduced. Cavusoglu et al. [9] described a multi-rate haptic interface method to introduce new forces into the haptic loop and to interpolate forces (at haptic rates) between force updates. We utilize this method, to in effect create a local approximation of the volumetric bone material on the client's system, for computing new forces at haptic update rates. Local haptic forces are computed within a client's *Multi-rate Haptics Renderer*. Equation 1 is the basis for the multi-rate haptic interfacing. A new collision force ($f(P_N)$) and a local force gradient ($\nabla f(P_N)$) are computed by the server based on the position of the haptic device at the time of calculation (P_N). P_n is the device's position at haptic update rates. A

new force ($F(P_n)$) is interpolated by the *Multi-rate Haptics Renderer* based on $\nabla f(P_N)$. 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.

$$F(P_n) = f(P_N) + \nabla f(P_N)(P_n - P_N) \quad (1)$$

The client's *Haptics Update Manager* can throttle the update timing based on how often it sends tool updates to the server. Similar to [9] we were able to calculate the force updates as low as 20 Hz using the multi-rate haptic interface. We found 20 Hz to be the lowest rate possible while still maintaining stability of the haptic interactions.

3.4.2 Bone drilling

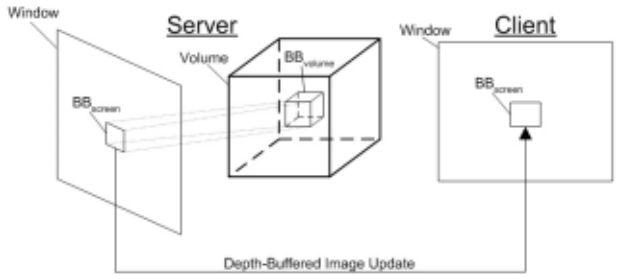

Figure 6: Optimizing visual updates by only sending a depth-buffered image of modified areas in the volume to clients. BB_{volume} tracked while drilling and used to compute BB_{screen}. Depth-buffered image for BB_{screen} captured and updated on client.

The bone drilling capabilities of the client's tools are computed separately from the haptic interactions to reduce the force calculation turn-around time from the server to the client. The server's *Haptic Interaction Simulator* places the tool information that is received from clients onto a *Tool Update Queue*. A *Bone Drilling Simulator* checks the queue for new tool updates and computes bone drilling between the virtual tool and the bone material. A local representation of the virtual tools is used to compute the bone erosion [4]. If the bone is modified while drilling, the server's *Graphics Renderer* generates a new DBI from the volume to update clients. Otherwise, no visual updates are sent and the next queued tool update is processed. To optimize the size of visual updates, only a DBI of modified areas in the volume are sent to clients. Figure 6 illustrates the update process. The *Bone Drilling Simulator* tracks a three-dimensional bounding box (BB_{volume}) of modified voxels within the volume. The BB_{volume}'s eight corner points (in object coordinates) are projected into screen coordinates to determine what area in the window is modified. A two dimensional bounding box (BB_{screen}) is found in screen coordinates with the min/max x-y values from BB_{volume}'s projected corner points. BB_{screen}'s coordinates are clipped to ensure they fall within the display window area. The DBI corresponding to BB_{screen} is captured and sent to the clients' *Graphics Update Manager*. The client's *Graphics Renderer* is notified when visual updates are received and updates the modified area within the *Depth-Buffered Region*. Since only modified areas are updated, we reduce the bandwidth requirements for synchronizing the clients. This also restricts expensive main memory to video memory transfers with *glDrawPixels* to small areas on the screen. It is worth noting that the size of BB_{screen} is affected by the virtual camera due to the perspective projection. The size of BB_{screen} becomes smaller as the field-of-view decreases and the volume's distance from the camera increases.

4 Experiments

Several experiments were conducted to evaluate the performance of the described hybrid update/rendering method and compare it with a volumetric update/rendering approach. In the later case, each user maintains a local copy of the C-VE and volumetric updates are used for synchronization. Volume rendering and haptic interactions are also computed locally. Two computers were used for testing. System 1 has a dual-core AMD Athlon 64 X2 4800+ processor, 2GB RAM, and two NVIDA GeForce7800 GTX video cards (256 MB) in SLI configuration. System 2 has dual Xeon 3.2GHz processors, 2GB RAM, and an NVIDIA Quadro FX 4000 (256 MB) video card. Graphics rendering is implemented with OpenGL [21]. When comparing the two systems, system 1 has greater computational and graphics rendering capacity than system 2. The two systems were connected by a 100Mbps local area network.

For the DBI results, system 1 was made the server and system 2 the client. The DBI that is sent to a client requires 4 bytes per pixel to store the RGB color and z-depth information.

The two systems were placed in a peer-to-peer configuration for the volumetric update method. The volumetric updates require 1 byte per voxel for bone density value updates. The receiving system updates the densities of the local volume, recomputes density gradient vectors (used for visual and haptic feedback), and updates volumetric textures used for volume rendering. Since system 2 is used as the test client for the DBI results, we mostly concentrate on system 2's results for comparing the two methods.

4.1 Visual results

We compared the rendering rates that are possible on system 2 using volume rendering versus the DBI-based hybrid rendering method. A window size of 500x500 and a screen resolution of 1024x768 was used to display a 128^3 volumetric VE. The DBIs were generated with a virtual camera 13" away from the volume and with a 60 degree field-of-view. The 128^3 volume is 300x300 pixels on the projected image. Without any interactions to the volume, 57 FPS were possible on average when rendering the volume locally on system 2. Other factors, such as the complexity of the volume and the volume's distance to the virtual camera, affect the volume rendering rates. For example, rendering a 256^3 volume dropped the rates to 26 FPS. Moving the virtual camera to 63" raised the rates to 91 FPS. In contrast, the DBI rendering rates are constant since the same size image is restored for the window at every frame and no other geometry operations are required. The hybrid rendering method allowed average update rates of 990 FPS (a 1,637% increase). This represents an upper bound since only the pre-rendered 2D image is restored.

Next, the possible update rates for updating a DBI versus updating the volume for different BB_{volume} sizes were evaluated. The results are based on system 1 updating system 2. The corresponding DBI sizes (BB_{screen}) for each tested BB_{volume} size is listed in Table 1. The width and height of the DBIs are the same since the volume was centered on the screen and was not rotated. From Figure 7 we see that the DBI method allows other users to be updated 1.5-2.0x faster than when updating the volume. Initially, the largest amount of time (87-90%) is taken by the volume rendering since the entire 128^3 volume is re-rendered for both methods. However, as the volume size increases the greatest amount of time shifts to updating the actual volume (e.g. computing gradients and updating textures). For example, the following results are based on updating a 80^3 volume size. For the DBI update method, 63% of the time is spent updating the volume, 21% transmiting/receiveing the update, 13% creating the DBI (1% for volume rendering and 12% for capturing the DBI), 3% updating the client's DBI, and $<$1% computing bone erosion. With the volumetric-based update method, 67% of the time is required to update the volume, 23% to transmit/receive the volumetric update, 8% rendering the update, and $<$1% to compute bone

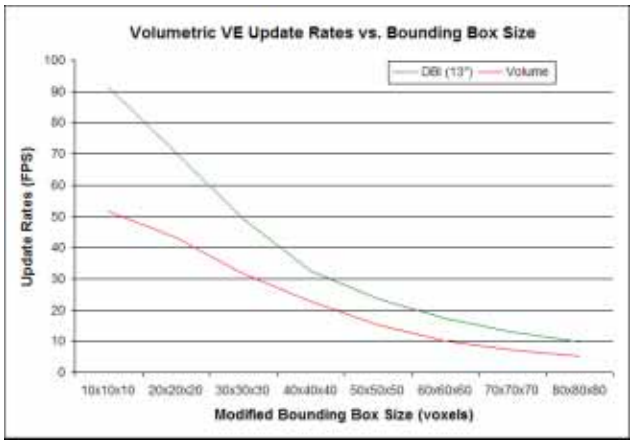

Figure 7: Update rates for different BB_{volume} sizes. See Table 1 for corresponding BB_{screen} sizes.

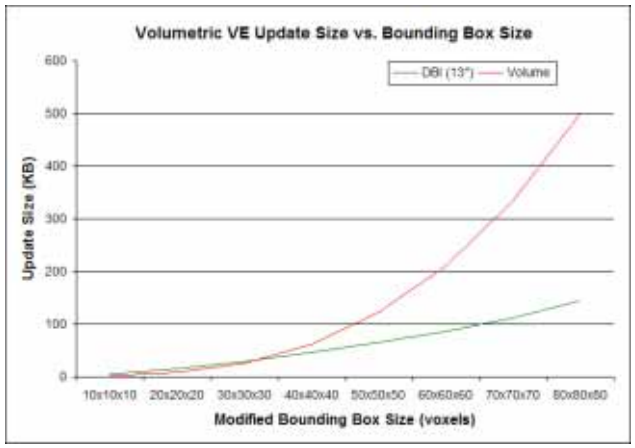

Figure 8: Size of updates for different BB_{volume} sizes. See Table 1 for corresponding BB_{screen} sizes.

Table 1: BB_{screen} size per BB_{volume} based on rendering the volume 13" from the virtual camera and a 500^2 window size.

Volume size (voxels)	DBI size (pixels)
10^3	36^2
20^3	63^2
30^3	87^2
40^3	109^2
50^3	130^2
60^3	148^2
70^3	170^2
80^3	193^2

erosion. Since the bottleneck for larger volume sizes is updating the actual volume, we see that the update rates for both update methods drop with larger volume modifications.

Finally, we compared the size of the updates that are generated by both methods. Figure 8 provides the results. Since only 1 byte per voxel is sent, the volumetric update size is N^3, where N is the dimensions of the volume. 4 bytes per pixel is required to send the RGB color and z-depth information. This results in a $4M^2$ sized update, where M is a DBI's dimensions. Based on these update size requirements and the BB_{screen} sizes in Table 1, we see that the DBI update size is slightly larger than the volumetric updates up to the third test volume update size. After this point, the volumetric update sizes grow much faster than the DBI updates. Despite having a larger update size for very small updates, we saw that the DBI update method provides significantly faster update rates on system 2. This is because the cost of rendering the volume is much more costly than updating and rendering the DBI. The larger volume sizes only have a little impact on the volumetric update rates because the bandwidth of the local area network is sufficient to handle the volumetric updates.

4.2 Haptic results

We evaluated the haptic update rates for simulating haptic interactions between the tools in Figure 1 and the volumetric bone. The multi-rate haptics interface method [9] described in Section 3 was used for both the DBI and volumetric update methods since system 2 cannot calculate haptic interactions for the perforator tool at 1kHz haptic update rates. On system 1, the haptic calculations took 0.56ms and 2.98ms for the bone drill and perforator, respectively. The bone drill calculations took 1.75ms and the perforator 9.19ms on system 2. Surprisingly, the DBI method allowed higher haptic update rates on system 2 than computing the interactions locally. This is because the network round trip only added 0.82ms to the very fast calculations of system 1. The rendering rates for the bone drill raised by 27% and the rates for the perforator raised by 142%.

5 Conclusion/Discussion

In this paper we describe a method to allow multiple users to interact with voxel-based C-VEs in real-time. We centralize the volumetric skull VE on a server to aid with synchronizing the C-VE. To overcome performance limitations of remote interactions and rendering, we describe a hybrid depth-buffered image (DBI) and geometry-based rendering method. Users only maintain a local DBI of the volumetric VE. The DBI is composited with locally rendered geometrical 3D models of patient anatomy and virtual tools. The tools can visually interact with the volumetric skull VE in real-time. Methods are also provided to allow the locally controlled virtual tools to interact with the remote volumetric bone VE. Both haptic and bone drilling interactions are simulated. A multi-rate haptic interface method [9] is used to overcome network latency to simulate remote haptic interactions. While drilling, only two-dimensional updates are required to update clients' DBI representation of the volumetric VE for synchronization. To optimize the bandwidth requirements for updating clients, we describe a method to determine what area of the rendered volume image is modified and only send the modified area. We also describe a method to correctly create a DBI using texture-based volume rendering.

For evaluation purposes, we also implemented replicating the volumetric C-VE on each user's machine and using volumetric updates to synchronize the local VEs. We found our approach to allow 1.5-2.0x faster update rates than the volumetric update approach. The update size for the volumetric updates can grow at a N^3 rate, while our method can grow at a $4N^2$ rate. Our 100 Mbps local area network is able to accommodate the larger volumetric update sizes of the replicated volumetic C-VE; however we expect that the larger updates will have a greater negative impact on the update rates over a wide area network. Additionally, the volumetric-based updates may quickly flood the network as the number of clients increases. Using smaller update sizes should allow our method to scale more gracefully than a volumetric-based update method. We plan to run additional experiments to test the scalability and to see how the two volumetric C-VE methods compare over a wide area network.

Since our C-VE is based on opaque voxelized bone material, we currently do not address semi-transparency. One possible approach

for supporting semi-transparency is to use a multi-layered depth image, where each layer represents a certain z-region within the VE. Each image layer can be rendered back-to-front and be alpha-blended with local geometrical 3D models. Another approach might be to use an attenuation function that is based on objects' materials to approximate semi-transparency.

The viewpoint does not change frequently in our application. However, if the view does change then the entire DBI needs to be updated. We are investigating methods to allow the viewpoint to be quickly changed by the client. The server could have a background process that renders cached copies of different views to quickly respond to a user's request to change the viewpoint. The client can also try to extrapolate new views based on the existing DBIs while the server generates a new view. Others have addressed issues similar to this.

Finally, the DBI is currently just used for visual synchronization of the different clients. It would be interesting to see if we can also use the DBI for computing local haptic response.

The greatest beneficiaries of our method will be thin client systems that lack the computational and/or rendering capacity to simulate the volumetric VE in real-time. Although our tested client is not a very low-end system, it still benefited from our method. Moving the volume and haptic rendering calculations onto the higher-end server PC raised the graphics rendering rates by 1,637%. The haptic rendering rates also improved by 27% and 142% for the bone drill and perforator, respectively. Freeing up additional computational resources will allow clients' to perform other work required by a surgical simultator.

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] http://oss.sgi.com/projects/ogl-sample/registry/index.html.

[2] http://www.sensable.com.

[3] Sgi opengl vizserver 3.5: visualization and collaboration. *Technical white paper. http://www.sgi.com/pdfs/3263.pdf*, 2005.

[4] E. Acosta and A. Liu. Real-time volumetric haptic and visual burrhole simulation. In *IEEE Virtual Reality (To appear)*, 2007.

[5] 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.

[6] M. Agus, A. Giachetti, G. Zanetti, and A. Zorcolo. Real-time haptic and visual simulation of bone dissection. In *IEEE Virtual Reality*, pages 209–216, 2002.

[7] J. Ahrens and J. Painter. Efficient sort-last rendering using compression-based image compositing. In *Second Eurographics Workshop on Parallel Graphics and Visualisation*, 1998.

[8] D. G. Aliaga and A. A. Lastra. Architectural walkthroughs using portal textures. In *IEEE Visualization*, pages 355–362, 1997.

[9] 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.

[10] K. Engel. Interactive visualization of volumetric data on consumer pc hardware. In *IEEE Visualization 2003 Tutorial, Basic illumination techniques*, 2003.

[11] K. Engel, O. Sommer, and T. Ertl. A framework for interactive hardware-accelerated remote 3d-visualization. In *Eurographics Data Visualization*, pages 167–177, 2000.

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

[13] C. Gunn, M. Hutchins, D. Stevenson, M. Adcock, and P. Youngblood. Using collaborative haptics in remote surgical training. In *WorldHaptics*, 2005.

[14] M. A. Hutchins, D. R. Stevenson, C. Gunn, A. Krumpholz, T. Adriaansen, B. Pyman, and S. O'Leary. Communication in a networked haptic virtual environment for temporal bone surgery training. *Virtual Reality*, pages 1–11, 2005.

[15] J. Kim, H. Kim, B. Tay, M. Manivannan, M. Srinivasan, J. Jordan, J. Mortensen, M. Oliviera, and M. Slater. Transatlantic touch: a study of haptic collaboration over long distance. *Presence: Teleoperators and Virtual Environments*, 13(3):328–337, 2004.

[16] M. R. Macedonia and M. J. Zyda. A taxonomy for networked virtual environments. *IEEE MultiMedia*, 4(1):48–56, 1997.

[17] K. Montgomery, C. Bruyns, J. Brown, G. Thonier, A. Tellier, and J. Latombe. Spring: A general framework for collaborative, real-time surgical simulation. In *Medicine Meets Virtual Reality*, 2002.

[18] 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.

[19] 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.

[20] D. Roberts and R. Wolff. Controlling consistency within collaborative virtual environments. In *IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 46–52, 2004.

[21] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, Fourth Edition*. Addison-Wesley, 2004.

[22] H.-Y. Shum and S. B. Kang. A review of image-based rendering techniques. In *IEEE/SPIE Visual Communications and Image Processing*, pages 2–13, 2000.

[23] H.-Y. Shum, S. B. Kang, and S.-C. Chan. Survey of image-based representations and compression techniques. *IEEE transactions on circuits and systems for video technology*, 13(11):1020–1037, 2003.

[24] M. Wimmer, M. Giegl, and D. Schmalstieg. Fast walkthroughs with image caches and ray casting. *Computers and Graphics*, 23(6):831–838, 1999.

[25] B. Wylie, C. Pavlakos, V. Lewis, and K. Moreland. Interactive rendering on pc clusters. *IEEE Computer Graphics and Applications*, 24(4):62–69, July/Aug. 2001.

[26] C. Xavier and M. Christophe. Pipelined sort-last rendering: scalability, performance and beyond. In *Eurographics Symposium on Parallel Graphics & Visualization*, 2006.