Volume Graphics (2006)
T. Möller, R. Machiraju, T. Ertl, M. Chen (Editors)

Capture and Review of Interactive Volumetric Manipulations for Surgical Training

Thomas Kerwin[†1,2] Han-Wei Shen[‡1] Don Stredney[§1,2]

[1]Ohio State University
[2]Ohio Supercomputer Center

Abstract

In this paper, we present a system to capture efficiently a given user's interaction with a simulation system involving the procedural removal of material inside a volume, and to allow for a full 3D, lossless reviewing of that interaction at a later date. We describe an extension of this system to enable reviewing to occur at arbitrary points in the surgical procedure. The extension uses a combination of volume snapshots and event recording to be efficient in both time and space requirements.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications J.3 [Computer Applications]: Life and Medical Sciences

1. Introduction

Simulations gain utility by the repeatability of training and testing procedures. Ensuring uniformity in conditions of a simulation gives a higher confidence to associations drawn between the performances. In addition, the uniformity and repeatability of procedures facilitates the review of individual performance. In training simulators, it is helpful to capture the interaction so that the user can evaluate his or her own mistakes or so the instructor can point out problem areas. In testing simulators where performance is used for evaluation, reviewing the interaction of the user with the system is critical for uniform, multiple evaluations and for the reproducibility of results. Key to consistent and precise evaluations is the ability of the system to return similar results under different conditions.

In interactive simulations, our experience is that while many novices prefer immediate notification of error, more proficient residents prefer to defer notification, and to receive feedback as a total score, perhaps with key milestones separately ranked. The simulation environment that we have designed requires capture of user performance for two contexts:

1. to capture an expert performance so that a novice may invoke a playback of how an expert would execute the procedure, and
2. to capture novice activities for an evaluation that can either be statistical or interactively viewed.

This medical simulator is being developed to integrate haptic, visual and auditory interaction in order to create a comprehensive simulation for the dissection of temporal bone for training purposes. The simulator uses an internal volumetric representation of the temporal bone located in the lateral region of the skull to emulate surgical interactions. The bone is represented in the simulator by a four channel RGBA volume. The user input for the bone drill is accomplished by using a Phantom$^{\text{TM}}$ three dimensional haptic force feedback device and a foot pedal that controls the drill speed. Visual feedback is accomplished with direct volume rendering. As the user drills into the bone, the bone density of the voxels is reduced and the display is updated with the new density values. For a detailed explanation of the architecture, refer to Jason Bryan et al. [BSWS01].

In this paper, we describe a system which provides capture and review of user modifications of the volumetric representation of a temporal bone. This system facilitates both of the

[†] kerwin@cse.ohio-state.edu
[‡] hwshen@cse.ohio-state.edu
[§] don@osc.edu

above contexts and does so efficiently in terms of both disk space and time. In addition, this system provides for more rigorous deferred feedback for the more experienced while not impairing immediate feedback for novices.

2. Related Work

As mentioned earlier, this work is an extension of the simulator by Jason Bryan et al. [BSWS01]. That simulator is designed to run on an SGI Onyx 2 workstation. Our revised system follows the same general architecture, but targets off-the-shelf PC systems with high-end graphics cards. Other temporal bone simulation environments have been developed by the Stanford BioRobotics Lab [MSB*04], Agus et al. at CRS4 [JTP*01].

Recordings of interactions can be used as data to other forms of analysis. Studies like those of Sewell [SMB*05] use derived quantities of the simulation such as visibility to quantify risk. Using a capturing system like ours would allow researchers pursuing similar ideas to do analysis without collecting more expert interactions. This would be impossible with a 2D movie.

One way to record the interaction is with a 3D movie. Extensive work has been done on various 4D volumetric compression schemes for accelerated playback. Sohn et al. [SBS02] use a lossy wavelet based algorithm with difference frames based on the general algorithm found in the commonly used MPEG movie standard and achieve compression ratios from 150:1 to 300:1 with root mean squared errors of around 0.10. Wang and Shen [WS04] extend the time-space partition tree described earlier by Shen [SCM99] with lossless wavelet compression to achieve compression results of 5:1 to nearly 20:1.

Unfortunately, we cannot exploit the substantial compression ratios obtained by lossy algorithms because the review of the interaction will be analyzed later by the instructors. Any misjudgment based on artifacts produced by lossy compression is unacceptable. Furthermore, many of the time series volume compression schemes in the current literature take considerable time to perform the compression. Nguyen and Saupe [NS01], using an accelerated algorithm, achieve compression times of around one to two minutes for a 256MB dataset. Because this simulator is designed to be used in an interactive environment, we want to avoid lengthy post-processing before review.

Additionally, using 4D compression techniques would require saving every frame of interaction as a separate volume and then processing the entire dataset at the end of the session. In our case, the volume is modified in specific amounts within a moving area of influence (the drill burr) rather than modified across the entire volume. By saving the input parameters of the drilling algorithm that modifies the volume, we can save much more space than a general method of saving raw data of the modified volume.

3. Capture and Review

The most obvious way of capturing a record of the user interaction is a movie capture. This is simple to implement: record a screen capture at the FPS desired. However, creating a two-dimensional movie of the simulation does not capture the full range of data. Arbitrary orientation of the dataset on review is impossible with this approach, as is using an interactive clipping plane to more precisely determine relationships between the surgical tools and critical anatomical structures. Making a fully three-dimensional movie by simply saving the volume data is not a viable option either. Although playback of large time series volumes is possible, recording volumes to disk in real time is not feasible, especially with sessions regularly lasting over fifteen minutes.

3.1. Our method

The solution is to record the events, rather than the outcome, by storing their position and time, along with all the information needed to recreate them. To review the original interaction, we use the same algorithms contained in the simulation logic to recreate the output from the user's input.

The events are categorized when they are captured. The categories that events fall under include drilling events, which save the speed and position of the drill, and state change events, like the changing of the burr tip size. We save these categorized events instead of raw device input to allow more easily the use of the devices during playback.

To rigorously compare the performance of different users on the simulation system requires a method to capture the procedure at the moment of use. The system therefore requires the abstraction of the user's activities. Our simulation system processes input from specialized input devices. Therefore, extension was relatively straightforward. We have employed a difference based system: instead of storing the state of the system at every timestep, we store the difference between timestep t_{i-1} and timestep t_i in terms of the user events that precipitated the data change between those two states. The general steps of this system are as follows:

1. User presses record button
2. Events are captured while user interacts with the system
3. User presses end record button
4. Event list is processed and snapshot volumes are created
5. Processed event list is saved for later review

3.2. Cueing

Cueing is an essential feature of a good capture and review system. It allows a reviewer to go to a specific time in an interactive session and start reviewing from that time step. Without cueing, the reviewer would need to watch the entire procedure multiple times when he or she is only interested in re-examining specific sections.

A potential drawback of the system described to this point is that when a cue point is selected, the system must, in effect, “fast forward” through the event list from the beginning to the time point specified by the user. The volume must be modified for every drilling event that interacts with the volume during review. Assuming drilling events will be distributed evenly in time throughout the sequence, the time it takes to cue to time T increases linearly: $O(T)$. Our system improves the performance to a worst case of $O(\log T)$.

We do this by using a sequence of snapshot volumes that can be used to jump ahead to defined points. The internal representation of the snapshot volume list contains pairs consisting of a filename and a time stamp. To start the playback from a time t, we need to find the right entry from the ones in this list (see figure 3), by finding the filename with the largest timestamp that is less than t. If none are found, then t is before any recorded snapshots and no more loading of data is needed. This operation can be done with a binary search, resulting in a $O(\log n)$ performance penalty, where n is the number of snapshot volume entries in the list. When the snapshot volume entry is found, we fetch the volume from disk and load it into memory.

After finding the snapshot volume x, we also need to find the events that, when replayed, will take the dataset from sequence time $time(x)$ to time t. This is a subsequence of the list E of all user events. This subsequence is $\{e_1, e_1+1 \ldots e_2-1, e_2\}$ where e_1 is the first element in E after the snapshot's timestamp and e_2 is the last event occurring before time t.

After finding these events and their positions in the list, we process all the events between them in chronological order: not in actual clock time, but as fast as the system can allow. The goal is to get the dataset to a replica of its state at time t so that the user can review the interaction in actual time starting from time t.

As mentioned earlier, this procedure takes only $O(\log t)$ time to execute: the binary searches are $O(\log t)$ and the time taken to replay the user events after the snapshot is loaded is bounded depending on the decision function discussed in the next section. In practice, the time taken to load the snapshot volume and play back the individual drilling events overwhelms the time taken to search the list, giving nearly constant time performance.

3.3. Snapshot capture and storage

An important component of a system that allows users to jump to arbitrary timestamps using snapshots is the procedure of taking and storing those snapshots. Our system is designed to run on standard PC systems, and capturing several gigabytes of data per simulation session can quickly become unmanageable in a typical medical teaching environment. Therefore, to store the snapshot volumes efficiently, we use both difference volumes and compression. Calculating both of these during simulation is too expensive and, in our tests, interferes with the interactivity of the simulator. This is unacceptable, so we need to do this in a short post-processing step.

Even with a limited number of volume snapshots, recording the entire volume every so often could add up to a large data size during a long interaction. To reduce the data storage needed, we use a difference volume technique. In the simulator, even though we have many voxels around the dissected area for spatial reference and realism, only a particular region in the volume is commonly modified. Consequently, we can save a great deal of space by encoding a snapshot volume by saving the difference between the current voxel opacity and the original opacity.

This encoding should not be confused with the type of difference encoding found in, for example, the MPEG standard. An example of this type of compression applied to volumetric movies is described by Guthe and Straßer [GS01]. In that case, the decoding of a frame x requires the information from frame $x-1$. In our case, we take differences from the dataset at timestamp 0. Therefore, in order to decode a difference volume at time g, we require just that difference volume and the original volume. If we had implemented the difference encoding for the volumes in a similar way as the MPEG standard, we would need to load many difference volumes to get to our initial cueing position instead of just one.

At the beginning of the interaction session, the system stores a copy of the volume for later comparison. The times that the snapshot volumes are taken are determined by a decision function. This decision function (call it D) must obey two requirements (also see figure 4):

$$D(t) > D(t-1) \quad \text{where there is no snapshot at t} \qquad (1)$$

$$D(t) = 0 \quad \text{where there is a snapshot at t} \qquad (2)$$

In other words, the function is non-decreasing until a snapshot volume is taken, at which time it resets. This is meant to be a measure of both the amount of difference from the last snapshot, and the time it would take to process events from the last snapshot. The goal is to choose a function that will decide to save snapshot volumes when it will take less time to load the difference volume from disk than it will to process the raw events in sequence to replicate that state from an earlier timestamp.

The decision function we use for this project is based on the number of voxels modified. For each drilling event, we calculate this number. If no voxels are modified, it takes very little time to process. If some voxels are modified, the burr kernel must be subtracted from an area of the volume, which is a significant performance hit for the simulator, and is the most demanding task besides rendering. Furthermore, the size of a difference volume is related to the modified voxels: if the number of modified voxels is too large, the difference volume loses compressibility. We compress the volume after differencing using the open source zlib library.

	Test 1	Test 2
interaction time	117	180
processing time	9.656	16.344
fast cueing	5	5.032
slow cueing	15	25.2

Figure 1: *Fast vs. slow cueing. Times are in seconds.*

4. Results

The development platform used for these results is an Athlon 64 3200+ with 2GB of RAM. The dataset used is a processed CT scan stored as a 128x256x512 four channel volume.

The use of the snapshot volumes decreases times when jumping ahead to a section of the interaction review by a significant amount, as seen in the figure below. In these tests, we gave the cueing algorithm the time index at the end of the interaction. The slow cueing row gives times for which the algorithm used the naïve approach where every event is played back. The fast cueing refers to our accelerated algorithm. Much of the time in the fast cueing is taken by the initial load of the volume, which takes about 2.5 seconds for both cueing algorithms. The time taken for decompression of the volume is only about 100 milliseconds.

The compression for the difference volumes is quite good. In test two above, five difference volumes were saved to disk with file sizes ranging from 123KB to 203KB. Given the uncompressed datasize of 16MB per volume, this shows compression ratios of 80:1. The file sizes increased as the timestamp associated with them increased. This is not surprising, since the difference from the original volume increases as the user drills away more of the bone. Still, most of the bone structure is untouched, generating many zeros and allowing very high lossless compression.

The size of the saved event list file ranges from 200KB to 500KB, and will increase as the time of interaction with the simulation increases. This is a required file for both types of cueing. As a comparison, a 2 minute long movie of a 3D interaction at 15 frames per second would have 1800 frames and would require more than 28GB of data uncompressed. We store the same information in one 500KB event file and five difference volumes of around 200KB each, totaling 1.5MB. This is equivalent to a compression ratio of more than 19000:1.

5. Conclusion and Acknowledgments

By capturing the user's interaction with the simulation, we can play back the interaction for expert reviewers and potentially use the data for automatic scoring systems and further research into drilling techniques and statistical analysis. Also, the framework we have described allows us to demarcate snapshots in the future with semantic meaning, giving a reviewer the option to skip to a stage in the surgery, rather than to a time four minutes from the start. The optimization metric would have to be altered to take into account logical separations in the procedure, either through explicit commands by the user or implicit information from heuristics or machine learning techniques.

This work is supported by grants NIH NIDCD 5R01-DC006458-02 and NSF RI-CNS-0403342. The authors would like to thank Dr. Gregory Wiet at Children's Hospital–Columbus for his continued support and feedback. Thanks also go to Irene Giller and Bill Kerwin for their helpful editing suggestions.

References

[BSWS01] Bryan J., Stredney D., Wiet G., Sessanna D.: Virtual temporal bone dissection: a case study. In *IEEE Visualization* (2001), pp. 497–500.

[GS01] Guthe S., Strasser W.: Real-time decompression and visualization of animated volume data. In *IEEE Visualization* (2001).

[JTP*01] John N. W., Thacker N., Pokric M., Jackson A., Zanetti G., Gobbetti E., Giachetti A., Stone R., Campos J., Emmen A., Schwerdtner A., Neri E., Franceschini S. S., Rubio F.: An integrated simulator for surgery of the petrous bone. In *MMVR* (2001).

[MSB*04] Morris D., Sewell C., Blevins N., Barbagli F., Salisbury K.: A collaborative virtual environment for the simulation of temporal bone surgery. In *Conference on Medical Image Computing and Computer Assisted Intervention* (2004).

[NS01] Nguyen K. G., Saupe D.: Rapid high quality compression of volume data for visualization. *Comput. Graph. Forum 20*, 3 (2001).

[SBS02] Sohn B.-S., Bajaj C., Siddavanahalli V.: Feature based volumetric video compression for interactive playback. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics* (2002), pp. 89–96.

[SCM99] Shen H.-W., Chiang L.-J., Ma K.-L.: A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *IEEE Visualization* (1999), pp. 371–377.

[SMB*05] Sewell C., Morris D., Blevins N., Barbagli F., Salisbury K.: Quantifying risky behavior in surgical simulation. In *MMVR* (2005).

[WS04] Wang C., Shen H.-W.: *A Framework for Rendering Large Time-Varying Data Using Wavelet-Based Time-Space Partitioning*. Tech. Rep. OSUCISRC-1/04-TR05, Department of Computer and Information Science, The Ohio State University, 2004.

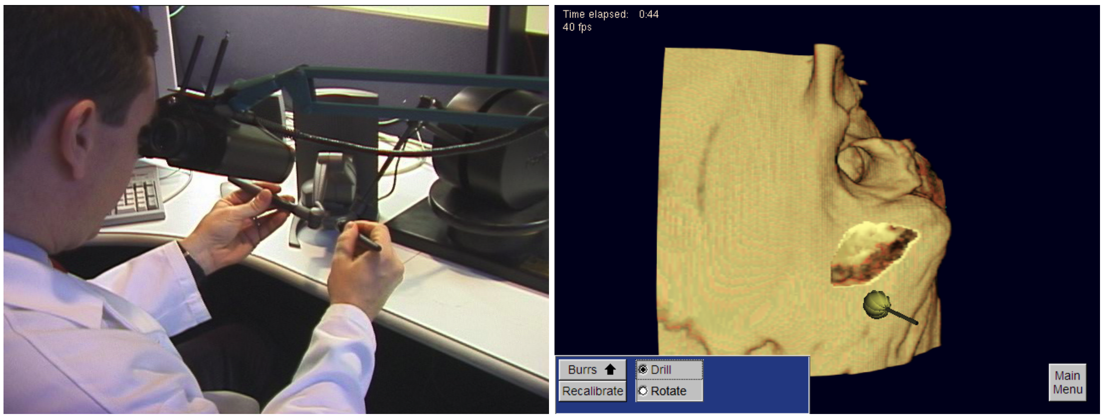

Figure 2: *Left: The simulator uses a binocular display for 3D stereo rendering and a 6 degrees of freedom haptic interface device. Right: A virtual mastoidectemy in progress.*

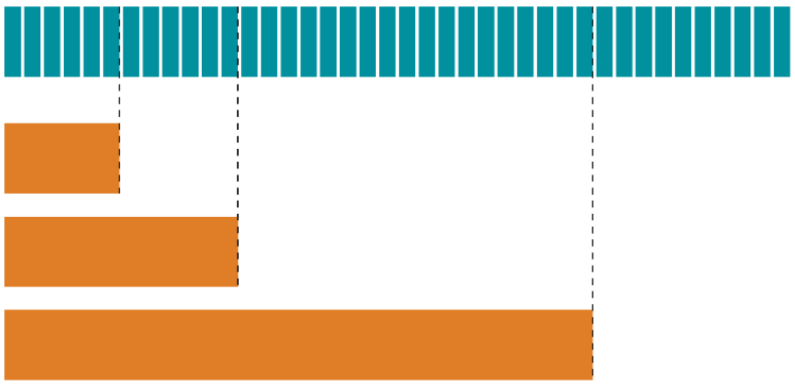

Figure 3: *Snapshot volumes (shown in orange) store the end result of applying a sequence of user events (shown in blue) from the beginning of the session to a specified point.*

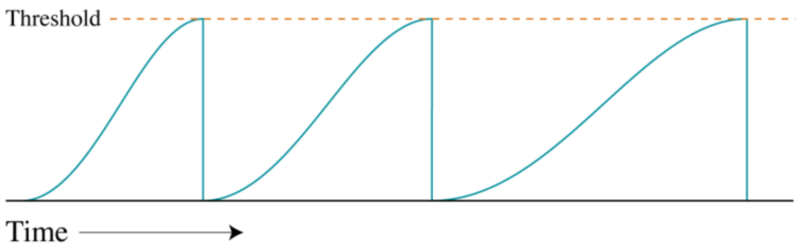

Figure 4: *Graph of the required behavior of D*