Visualization of 3-D Wave Propagation in the Heart – A New Technique

Han-Wei Shen, Dept. of Computer Science, hwshen@cs.utah.edu
Prasad B. Gharpure, Dept. of Bioengineering, gharpure@vissgi.cvrti.utah.edu
Christopher R. Johnson, Dept. of Computer Science, crj@cs.utah.edu
University of Utah, Salt Lake City, Utah

Abstract

We present a new method, called *differential volume rendering*, to speed up the process of volume animation for flow visualization. Data coherency between consecutive simulation time steps is used to avoid casting unnecessary rays from the image plane. We illustrate the utility and speed of the differential volume rendering algorithm with simulation data from electrical wave propagation within a cellular automaton model of the ventricular myocardium. We can achieve considerable disk-space savings and nearly real-time rendering of 3-d flow using low-cost, single processor workstations.

Introduction

We have developed a cellular automaton model to study electrical activation in the myocardium. Each element in the model simulates a region of the ventricular myocardium and has the following characteristics: excitability, cycle length dependant absolute refractory periods and relative refractory periods. Anisotropy of impulse propagation from one element to another is introduced by a dependence of the propagation velocity on the local fiber orientation, with the fastest conduction oriented along the fiber axis. Details of the model may be found in [1] . The model is primarily used to study factors that affect the vulnerability of the heart to fibrillation. Hence, most observations need to be concentrated on the propagating activation wavefront. Volume visualization was chosen to be the most suitable means of visualizing the 3-d nature of the wavefront throughout the entire myocardium. The major drawback of conventional volume visualization is that it is extremely resource (time and memory) intensive. We have exploited the temporal coherence in simulated data to achieve a significant speedup over traditional volume rendering schemes. To ensure numerical stability, most flow simulations are computed using small time steps such that a very small fraction of the total elements change their values from one step to the next. If we concentrate our rendering efforts solely on the changed cells, the number of rays that need to be cast decreases dramatically and yields a sizable reduction in terms of the amount of time devoted to rendering. Since we concentrate on the differences in data throughout the simulation, we call this new technique - *Differential Volume Rendering.*

Differential Volume Rendering

From preliminary studies of our wave propagation simulations, the only elements which changed values between consecutive time steps were the activated cells and their neighbors. In addition, when a sequence of propagating images is animated, the viewing parameters usually don't change. Thus, the pixels in the new image will keep the same color values unless they corresponded to changed data elements. The main idea of the differential volume-rendering algorithm is to exploit the temporal coherence between sets of volume data. Rays are cast along a path corresponding only to changed data elements. By decreasing the number of rays that need to be cast, and retaining the color values from the non-changing pixels , a significant amount of time saving can be achieved for volume animations.

Visualization Pipeline

The visualization pipeline of the differential volume rendering method can be divided into two phases, *static* and *dynamic*. In the static phase, operations are performed once for a data set. Data is generated from a simulation which might typically consist of hundreds or thousands of time steps worth of information. The simulation data of consecutive time steps is then compared to obtain the positions of changed data elements. The positions of those changed elements and their corresponding time step values are output into a single *differential file* which is the only information needed to produce animated volume rendered images. Due to the small fraction of variation between consecutive time steps, the differential file which replaces the whole sequence of volume data yields tremendous saving of disk space.

In the dynamic phase, the positions of changed elements are extracted from the differential file at each time step and the pixels where new rays need to be cast are computed according to the viewing direction. The resultant pixel positions are placed into a ray casting list, which is referred to by the ray casting process before firing new rays to produce the updated image. These operations are classified as the dynamic phase because the pixels corresponding to those changed elements are dependent on the viewing direction. The pixel positions remain undetermined until the user specifies the viewing parameters.

Results and Discussion

The software for the differential volume rendering algorithm has been implemented using OpenGL, which is portable across most hardware platforms. The parallel projection and template-based 26 connected discrete ray casting paradigms [2] were used in our implementation. The performance measurements were evaluated on a SGI Indy workstation with a single 100 MHZ MIPS R4000 processor.

Upon analysis of our simulation data, we noticed that only a small fraction of elements changed states between time steps. Table 1 compares the average rendering times for a differential volume renderer and a traditional volume renderer using a 128 x 128 x 128 volume. The rendering times for the first image are the same, with a significant reduction for subsequent time steps. In our simulation data, the average time to render 100 images without adopting the differential method was approximately 13.39 x 100 = 1339 seconds. Differential volume rendering achieved the same amount of rendering and the same image quality in only 65.5 seconds, a savings of 95%. The disk space taken by the 100 time steps of volume data was 2.10 x 100 = 210 M Bytes. By using the differential volume rendering algorithm, which only require the volume data of the first simulation time step and the differential file, we could reduce the allocated disk space to 2.1 + 2.08 = 4.18 M Bytes, a reduction of 95%.

The robustness of the differential volume rendering was tested by varying the number of changed elements between time steps. Table 2 lists the rendering times for different amount of changed elements in a 64 x 64 x 64 volume. When the percentage of changed elements was over 50-60%, the performance of the differential volume rendering algorithm became worse than the regular ray casting method, due to the overhead needed to calculate the pixel positions before casting new rays.

Although our algorithm has limitations when the number of changed elements exceeds 60%, for most flow visualization applications the number of changed elements during the propagation at each time step constitutes only a small fraction of the whole volume. Therefore, differential volume rendering represents an attractive technique for flow visualizations.

Currently we are in the process of evaluating and adopting different rendering paradigms as well as parallelizing the algorithm to maximize performance.

Time Steps	Regular Ray Casting	Differential Ray Casting		
		Pixel Calculation	Ray Casting	Total
0	13.399	0	13.399	13.399
10	13.398	0.131	0.315	0.446
20	13.389	0.184	0.549	0.733
30	13.347	0.209	0.671	0.880
40	13.315	0.211	0.713	0.924
50	13.377	0.189	0.618	0.807
60	13.390	0.135	0.287	0.422
70	13.401	0.118	0.173	0.291
80	13.397	0.112	0.153	0.265
90	13.398	0.105	0.129	0.234

Table 1: Rendering time (in seconds) at selected time steps using both the regular ray casting and the differential ray casting algorithms

% of Changed Elements	Differential Ray Casting		
	Pixel Calculation	Ray Casting	Total
Regular Method	0	2.372	**2.372**
5%	0.151	0.127	0.234
15%	0.482	0.062	0.544
25%	0.643	0.341	0.984
35%	0.898	0.483	1.381
45%	1.142	0.585	1.727
55%	1.390	0.713	2.103
65%	1.642	0.827	**2.469**
75%	1.892	0.941	2.833
85%	2.140	1.070	3.210
95%	2.383	1.149	3.532

Table 2: Rendering time (in seconds) for different amount of changed elements in a 64 x 64 x 64 volume

References

[1] P. B. Gharpure and C. R. Johnson. A 3-dimensional cellular automaton model of the heart. *Proceedings of the 15th annual international conference of the IEEE EMBS*, pages 752–753, 1993.

[2] R Yagel and A. Kaufman. Template-based volume viewing. *Computer Graphics Forum*, 11(3):153–167, September 1992.

[3] P.B. Gharpure, H. W. Shen, and C. R. Johnson. Visualization of scalar and vector fields in a cellular automaton model of the heart. Technical report, Dept. of CS, Univ. of Utah, 1994.