A System for Large-Scale Visualization of Streaming Doppler Data

Peter Krištof
Microsoft
pkristof@microsoft.com

Bedrich Benes
Purdue University
bbenes@purdue.edu

Carol X. Song
Purdue University
cxsong@purdue.edu

Lan Zhao
Purdue University
lanzhao@purdue.edu

***Abstract*—The NEXRAD Level II super resolution Doppler radars continuously scan the atmosphere above the continental USA, providing a stream of temporally and spatially misaligned large volumetric data about cloud reflectivity, wind velocity, and spectrum width. This data is used for immediate and long term weather predictions. However, because this large amount of sparse streaming data is not temporally aligned, the existing approaches rely either on a 2D projection of the 3D data, or the display of the 3D data only for a single radar. We present a framework that enables users to interactively access, analyze, and visualize the Doppler reflectivity data directly in 3D for multiple radars. Our approach extends the existing body of work on large-scale storage of global weather data and out-of-core volume rendering using CUDA ray-casting. The asynchronously streamed reflectivity data from multiple radars are first temporally aligned and then processed to a hierarchical format that is suitable for a large-scale volumetric visualization in near-real time with a minimal run-time processing. This approach also allows for varying precision and level of detail.**

***Keywords*-Volumetric Visualization, GPU, CUDA, Hierarchical Data Structures, Large Scale, NEXRAD.**

I. Introduction

Weather affects everyone on our planet and weather forecasting helps reduce the effects of weather by providing timely alerts. Today's weather forecasting uses modern technology, of which the Weather Surveillance Radar-1988 Doppler (WSR-88D) radars is probably the most important.

The WSR-88D is a pulsed Doppler radar used to monitor meteorological and hydrological phenomena [1]. It continuously scans the air and provides 3D discrete fields of: reflectivity (precipitation), radial velocity (wind), and Doppler spectrum width (turbulence). Data is provided through the Next Generation Radar (NEXRAD) network that includes more than 150 Doppler radars at approximately $230km$ spacings across the continental US (Figure 1).

The NEXRAD data is 3D and time varying, however their format, acquisition and the time dependence make them difficult to process and visualize. One issue is their spatial sampling irregularity: due to varying overlap of radar scans, radars operating either in coarse or dense sampling mode (normal and precipitation mode), and locations closer to the radar being sampled with higher density. The data is therefore misaligned and non-uniform in a global geographic space. Furthermore, actual raw data usually contains ground clutter or false echoes that should be removed before visualization. Finally, the last issue is the size of the data, which is provided in a compressed form with a compression factor about ten and a stream rate of 40 MB/minute for the entire NEXRAD network.

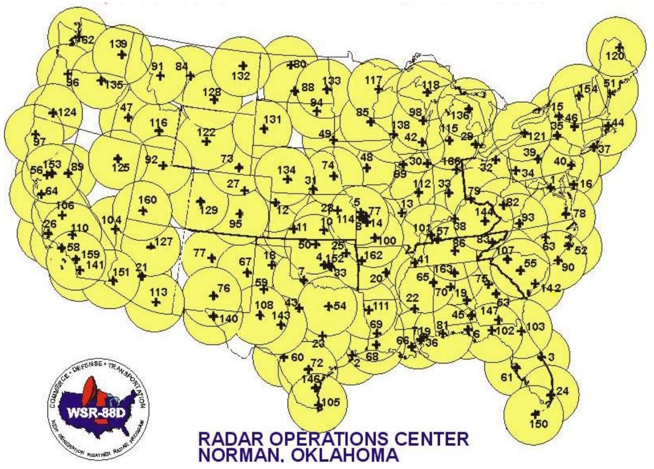


Figure 1. NEXRAD Coverage of the USA (courtesy NOAA).

We have developed a framework that allows for near real-time visualization, processing and delivery of NEXRAD data. Our system continuously retrieves and processes the radar data. Using TeraGrid's computational resources, we process the data from all radars, synchronize them in time, and align them geospatially. All data is then combined into a three-level hierarchical data structure suitable for large-scale visualization. The highest level is a geospatial quadtree storing a forest of octrees in the second level. This top level, motivated by the thin-layered 3D coverage of the scanned weather data, is used for a quick localization of the 3D volumetric data stored in the second level represented by octrees. The third level of our data structure is stored in each octree leaf and is a small 3D grid of predefined size. These "bricks" are processed directly on the GPU allowing for fast full volumetric visualization using CUDA ray-casting. Figure 2 shows an example of volumetric visualization of eastern coast of the USA.

The main contribution of our work are:

1) The design of a unified geospatial data structure for an efficient GPU-oriented displaying of the high-resolution 3D reflectivity data from multiple radars
2) A new algorithm for adaptive spatial and temporal

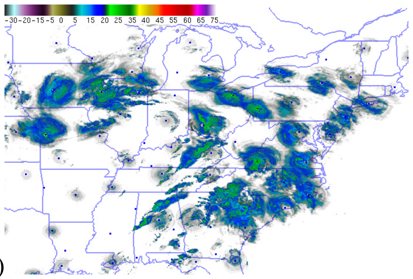
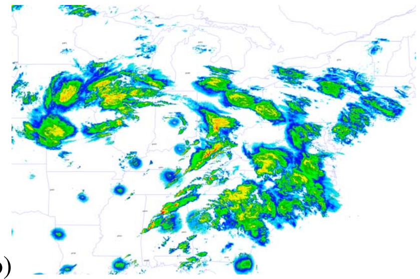
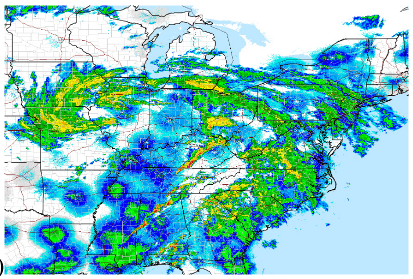
a) 
b) 
c) 

Figure 2. Volumetric a), maximum intensity projection b) and corresponding NWS's 2D visualization c) of reflectivity data from NEXRAD Doppler radars over eastern United States at 03:20 GMT on 04/25/2010. The timestamp on NWS image is 03:18 GMT.

alignment of the heterogenous radar data.

3) A new visualization algorithm for NEXRAD volumetric datasets.

II. Related Work

Doppler radar data is generated in the local radar's spherical coordinates. Data from multiple radars needs to be combined as shown in [2] where the data was mosaicked into a single 2D projection and transformed into the geographic space [3]. However, the reflectivity data have nonuniform density and the conversion to geographic coordinates results in an inconvenient conical structure, as shown in Figure 3. This is due to beam spreading, which makes the reflectivity sampling rate higher at the base of the radar and smaller with an increasing range. The missing data between elevation scans can be generated using one of the interpolation schemes for conversion of radar reflexivity data into cartesian coordinates [4]. However, this approach discusses data conversion from a single radar and do not address merging the data from multiple radars that is the focus of our work.

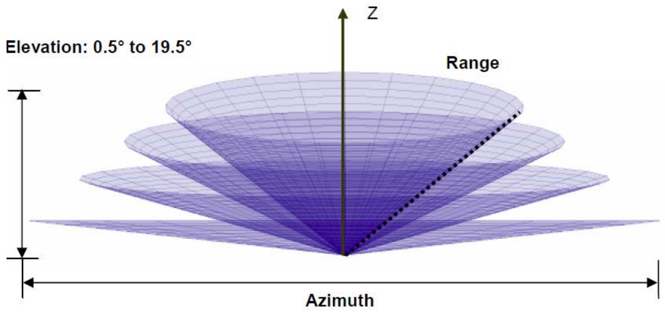


Figure 3. WSR-88D volume scan structure.

3D data completion from multiple radars using several interpolations was addressed by various researchers. Zhang *et al* [5] analyzed four interpolation schemes, summarized the choice of the interpolation to be application-dependent and suggested vertical interpolation for convective storms and both vertical and horizontal interpolation with distance-weighted mosaicking for general Cartesian grid mapping. Xiao et al. [2] evaluated a method using linear interpolation in all three dimensions and concluded the vertical interpolation with the nearest neighbor mapping on the range-azimuth plane provides results similar to the raw data.

Furthermore, the structure of the Doppler radar scan has some other inherent problems such as beam height increasing due to the curvature of Earth and missing data acquisition below the lowest beam (e.g., at angle $0.5°$) and above the highest beam (e.g., at angle $19.5°$), which is referred to as the ”cone of silence”. These issues can be alleviated to some extent by combining the radar data from multiple sites of the NEXRAD network and utilizing the scan overlap of neighboring radars.

Reflectivity Visualization is most commonly provided by simple 2D projection of the 3D reflectivity data. The projected value can represent either a base reflectivity, which is a value from a single elevation scan, or a composite reflectivity accounting for the strongest reflectivity from any elevation angle. However, the data may contain additional information, such as formation of deep convection or a forming tornado that is lost by the projection. The volumetric visualization of Doppler radar data has predominantly been done only for a single radar, because of the different access to the data, its size, and the spatial overlap. Specifically, this has been done by a 3D visualization of Doppler reflectivity using either isosurfacing or the texture slicing method to visualize volume from a single radar.

Reflectivity combined into a pre-defined 3D rectilinear grid and displayed using the texture-slicing with a post-interpolative transfer function on the GPU was used in [6]. The 3D grid is created by merging the radar data from radar sites that are within the user selected area. This approach suffers from aliasing caused by undersampling and poor data resolution since the size of the structure, is limited by the available GPU memory, and has a low time performance.

To render the reflectivity volumes from all the radars interactively, using a 3D grid representation is not efficient, considering that the weather data is usually sparse and large. A grid of reflectivity values covering the whole continental U.S. at the resolution equal of the highest resolution of super-resolution scans by Doppler radars would require over a hundred of terabytes of data. This is due to the conical structure of volume scans, in which the reflectivity samples can be as close as few meters or maybe even closer considering the irregular overlaps of the radar scans.

An interactive exploration of the radar data was presented by [3]. The authors applied a level-of-detail (LOD) technique by adopting a multi-level hierarchical data structure that matches the geographic nature of the thin coverage of reflectivity data around the globe. This work was extended in [7] by developing a framework for capturing multiple time steps and merging the radar date with satellite imagery.

Large-scale Volumetric Visualization is based on direct volume rendering (DVR) and it has been a subject of active research for a very long time. It can be classified into two main categories: object-order methods (splatting, shear-warp, 3D texture slicing) and image-order methods (e.g., ray-casting). We refer the reader to an overview of real-time volumetric visualization techniques in [8].

Interactive methods for large-scale volume visualizations have become possible with the introduction of programmable graphics hardware. With the advance of the computer graphics hardware and application of LOD techniques and multi-resolution octrees or N^3 trees it was shown that the volume rendering of large data sets can now be done in real-time on current GPUs [9]. The authors presented an efficient streaming of data to the GPU with a low computational demand on the CPU. The streaming is guided by the information computed during ray casting. Similarly, [10] utilized N^3 trees to encode a large astronomy data set consisting of hundreds of GBs and visualized it with image-space aligned splatting. Visualization of even larger datasets was shown by [11]. The authors proposed a visualization driven 3D data construction to render petascale microscopy data.

The remainder of our paper is organized as follows: we continue with Section III that provides a high-level overview of the solution and then the following section describes the data processing pipeline. Section V discusses the visualization part and the last two sections show results and conclusions with some thoughts about the possible future work. Appendix describes details about the NEXRAD data and its delivery.

III. System Overview

The system overview is depicted in Figure 4. The NEXRAD radars produce air scans in an asynchronous mode. Each complete radar scan (see Figure 3) is transferred to the local repository on the Purdue University TeraGrid system [1]. The data is the preprocessed and synchronized into regular time intervals. We use 10 minutes that is the time required for a complete scan of the sky generated by a radar. The data from varying time stamps are linearly interpolated in time to these intervals. This is done for each scanned ray separately as the different rays have different time stamps caused by the radar rotation.Next, the synchronized radar data is combined into a global multi-resolution hierarchical data structure and visualized with the GPU-oriented rendering algorithm.

The radar reflectivity data is stored in a hierarchical data structure in geographic coordinates (lat/lon). The Doppler radars scan as far as $230\mathrm{km}$ at the highest elevation of 19.5° while the altitude is sampled merely up to $75\mathrm{km}$. This makes the data coverage very sparse in the altitude dimension around the globe. Therefore, we chose to use a multi-level hierarchical data structure.

Our data structure can adapt to the input data and focus the work of processing and high-resolution sampling merely on the non-empty regions. Nodes, storing a constant-sized 3D grids, are used for storing the volume data to utilize hardware-accelerated data filtering during rendering.

IV. Data Delivery and Processing

The objective of this work is to allow a fast interaction and visualization of Doppler data over the entire USA. Currently, the multi-grid nature of sparse radars with varying coverage makes this task difficult. We have decided to use a hierarchical data structure that is suitable for this purpose.

A. NEXRAD Data Delivery

The NEXRAD data is continuously streamed from the National Weather Service (NWS) using a Local Data Manager (LDM) developed by Unidata. The radar data is stored and managed by an iRODS (Integrated Rule-Oriented Data System)-based data grid supported by the Purdue TeraGrid resource provider.

B. Multi-resolution Hierarchical Data Structure

The data hierarchy is depicted in Figure 5. The first level is formed by a lat/lon quadtree, which is refined until the lat/lon sizes are of the same magnitude as the altitude dimension and there is at least one radar within radars range span. This way each non-empty quadtree leaf represents a nearly cubical volume space. The second level is an octree formed in lat/lon/alt dimensions for each non-empty quadtree leaf. Similarly, each octree leaf connects to the third level by storing a pointer to a *brick*, which is a small 3D regular grid of predefined size M^3 and M is the size of one dimension. They correspond to the highest resolution of the stored volume data. The small constant-sized 3D grids allow for fast grid-based ray casting and are suitable for hardware accelerated data visualization [9].

To sample various resolutions during the visualization all the tree nodes have a brick associated with them as well. These bricks represent lower resolutions of the volume data and are built by averaging the bricks of the node's children. The resolution of the octree leaf is determined by the brick cell that must contain at most one reflectivity sample. This value is estimated by analyzing the data resolution locally

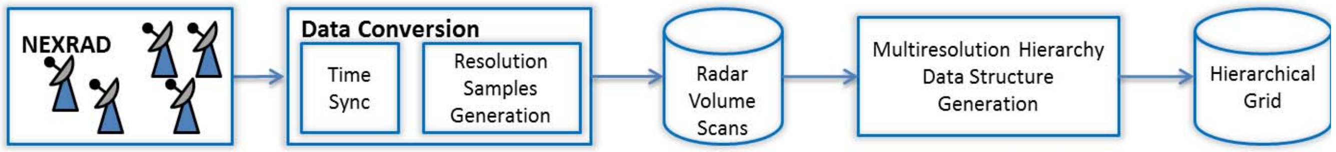

NEXRAD
Data Conversion
Time Sync
Resolution Samples Generation
Radar Volume Scans
Multiresolution Hierarchy Data Structure Generation
Hierarchical Grid

Figure 4. System overview. The raw data produced by hundreds of NEXRAD radars are collected at the TeraGrid system and georeferenced. Data sets are then synchronized in time and converted into preprocessed volume scans. The (partially overlapping) volumes are then merged into a 3D hierarchical structure.

around each reflectivity sample. In particular, the resolution is computed as the geographic distance to the nearest neighbor that has a different reflectivity value than the source sample. The resolution information is precomputed for each non-zero reflectivity sample and we refer to these samples as *resolution samples*.

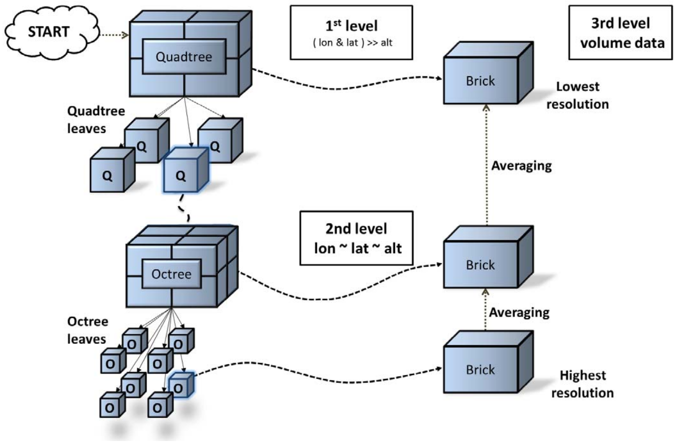


Figure 5. Multi-resolution hierarchical data structure for reflectivity data.

We store the tree nodes in a 3D texture, referred to as the *node pool*. The nodes are then accessed through a 3D texture index on the GPU. The node data are repartitioned to occupy only 64 bits so as to lower the memory requirements and improve the data coherence, which in turn helps the texture caching. The bit-wise structure of a node is described in Table I. Each node either contains a brick or a single data value and stores a pointer only to the first child. The remaining children are stored right after the first child, so that no more pointers are necessary.

Bits	Description
29	pointer to the first child
1	node/leaf
1	brick (octree for quadtree leaf) has been loaded
1	The content is a reflectivity value or described by a brick
18	single reflectivity value/brick (octree for quadtree leaf) idx
7 & 7	min. & max. reflectivity value of the subtree

Table I
BITWISE REPRESENTATION OF THE 64BIT TREE NODE.

C. Data Structure Building

The raw radar data is first synchronized in time. The radars operate at different time intervals and different scanning speeds, so data from each radar is interpolated in time to provide data at the same time stamp.

The quadtree is constructed over all the radar sites and each quadtree leaf stores a reference to radars that are within the radar radius from the leaf. Next, we build the octrees and their bricks. An octree is built using all the resolution samples that are within its bounding box. We subdivide the octree until the maximum data resolution for a brick has been met. The data resolution is evaluated by finding the resolution sample with the shortest distance. Then we build a brick for the leaf by computing reflectivity values for each cell from all the contributing radars. After all the leaf bricks are computed, the bricks for parent octree nodes are built by averaging the children's bricks in a bottom-up fashion. This process is then repeated for the quadtree nodes after the all the octree nodes have been built.

The most computationally demanding part is building the leaves' bricks. For each brick, we have to compute reflectivity values from contributing radars for M^3 (i.e., for $M = 32$ that is $32,768$) brick cells. The $M = 32$ was found experimentally as a compromise between the occupancy of the GPU memory and frequency of cache hit/misses. The computation per brick cell include converting the cell's geographic position to the radar's spherical coordinates (computed by using several computationally expensive trigonometry and squared root functions) and interpolating the neighboring values using the vertical interpolation with nearest neighbor mapping in the azimuth-range plane [2].

The bricks are compressed using Run-length encoding (RLE) that has been proven to be efficient for compressing 3D memory blocks of reflectivity data [6].

V. VISUALIZATION

Our visualization system is a large-scale GPU ray-guided ray casting that employs *out-of-core rendering* thereby allowing for visualization of datasets larger than system or GPU memory in a manner inspired by [9]. The main difference between their approach and ours is that we apply the ray-guided GPU ray casting to a two-level (Quadtree and Octree) tree data structure that is more appropriate for atmospheric data. Also, we extend on empty space skipping by analyzing the transfer function and min/max values in the tree nodes.

The data management is guided by the Least Recently Used (LRU) algorithm on the CPU. The LRU table is updated with the run-time information (i.e., tree node's brick usage) collected during GPU ray casting. The rendering pipeline is depicted in Figure 6.

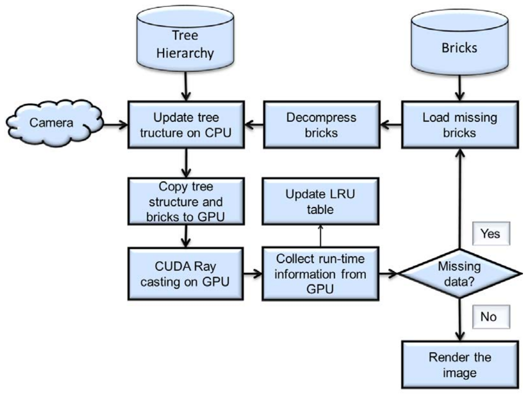


Figure 6. Visualization overview.

The GPU processes multiple rays in parallel. Each ray is initialized per respective pixel and then it is traversed until it requires a brick to be loaded or it leaves the volume as described by the following algorithm.

1) **Traverse the tree from the root of the quadtree**. The tree hierarchy is traversed until the node providing the desired resolution for the current position p is found. That is including any traversal through an octree after reaching a quadtree leaf.
2) **Transform the ray into the node space.** A ray's direction d and position p are converted into position p_B, relative to the node's brick space, so that $p_B \in [0,1]^3$.
3) **Perform volume ray casting in the brick.** The ray is cast through the $[0,1]^3$ volume of the brick. The colors and the opacities are integrated along the ray using a user-defined transfer function.
4) **Update the ray position.** The integrated distance in the brick is converted to the tree root's space and the ray's position p is updated. This position then serves as the input position to the next iteration.
5) **Check for the ray termination.** The ray is terminated when it leaves the tree's volume or when the opacity reaches one. Otherwise continue with step 1.

A. Tree traversal

The ray traversal of the hierarchical data structure is initiated in the tree root using the *kd-restart* algorithm, which, once it finds a node, starts back at the root on the next query. This traversal is efficient because the point coordinates p can be directly used to locate it within a node [9]. Let $p \in [0,1]^3$ be the point's local coordinates in the quadtree's bounding box, c be the pointer to the first child of the root and let's assume the tree hierarchy is stored in a 3D texture. The offset to a child, to which the p falls, is $\lfloor 2p \rfloor$ for p_x and p_y coordinates within a quadtree node and p_x, p_y and p_z within an octree node. Then, the pointer to the child is simply $c + \lfloor 2p \rfloor$.

The descent is iterated until either a leaf or a node with the desired resolution is reached i.e., one voxel projects to at most one pixel. If the node represents a single color, the volume integral is computed analytically. Otherwise, the node has a brick associated with it and the standard ray marching is applied. It should be noted that the ray direction d changes as we descend within the quadtree because only the x and y dimensions are subdivided. In an octree, where all dimensions are subdivided at the same time, the d is constant. The integrated distance, which is expressed in the node's local space $[0,1]^3$, for the ray is transformed into the quadtree's root local volume space and the p is moved according to that distance along the ray. The new p is then the input to the next descent.

B. Brick Caching

The ray casting allows for out-of-core rendering. Specifically, during a ray casting pass we collect information on which nodes have been used or need to be loaded. The tree hierarchy is small enough to be kept in the memory. However, the bricks, which store the actual volume data, take up to tens of gigabytes in uncompressed format and have to be loaded only when they are required. For this purpose, we keep an array of flags for each node. The flag can have one of the following three values: First, the node was not reached during ray casting. Second, the node was reached, but the brick is missing from the working set on the GPU. When this happens during ray casting we terminate the rays traversal for the current frame. Because of this, it can take several frames to reach a fully ray-casted image Third, the node was reached and the brick is available. After the ray casting is finished, we copy the 2D array of node flags to the CPU and update the LRU table accordingly. If the node was visited, its priority is increased in the LRU table. The missing bricks are loaded and if the brick pool is full we remove the bricks with lowest priorities from the LRU table.

Both the *node pool*, which contains the all the nodes of the tree, and the *brick pool* on the GPU are stored as 3D textures on the GPU. This improves the locality of the data on the GPU and makes the caching more effective.

C. Performance strategies

By using hierarchical data structure we do not trace the empty space as it is not refined in the empty regions. Moreover, we can skip the regions with data that is of no interest to us as well. Such data is defined by setting its opacity to zero in the transfer function. Thus, we can use the transfer function to speed up rendering by skipping the subtrees that do not contribute to the final image. We added min/max reflectivity variables to the node structure (shown in Table I) accounting for the lowest and maximum reflectivity within the subtree. During the tree traversal step, we sample the transfer function using the min/max values of a node and if the integral is zero we can safely skip the area.

VI. Results

A. Preprocessing performance

The system was tested on a high resolution data set from 116 radars that produce around 10GB of data every ten minutes depending on the weather conditions. After the data was delivered from TeraGrid network, resolution points were generated for each radar and then the hierarchical data structure was constructed. The critical value for the actual tree refresh is ten minutes; that is the interval for which the radar data is polled (one entire scan of the radar).

	Single CPU	OpenMP	OpenMP + CUDA
Resolution points	$417s$	$56s$	-
Build tree	1999s	$502s$	$149s$

Table II
PREPROCESSING TIMES FOR CONSTRUCTING THE DATA STRUCTURE FROM 116 SITES AT 8:50 AM (GMT), 4/24/2010.

The performance results of processing the high resolution data set are summarized in Table II. We performed the preprocessing on a quad-core i7 930 desktop machine with 6GB of memory and an NVIDIA 480 GTX card.

The single-thread processing took over 40 minutes, from which 33 minutes were spent on building the data structure and 7 minutes were spent creating resolution samples for all radars. The output bricks were compressed using RLE and the time step was stored at 2.5GB of compressed data (over 10 GBs of uncompressed data).

To fit the processing into the 10 minute interval, we multi-threaded the code using OpenMP and CUDA and using 8 logical threads. Applying only OpenMP processing reduced the time to 9 and a half minutes. By offloading the brick processing to the GPU, we brought the total preprocessing time down to 205 seconds, leading to nearly 12× speedup. The tree construction has essentially two parts - quadtree and octrees construction. The main bottleneck is constructing octrees and the bricks in their leaves. Building of octrees were directly spread among OpenMP threads. On top of that, this is sped up by each OpenMP thread submitting a brick for GPU processing for each octree leaf. Although there is still room for improvement, especially in smarter data fetching from the hard drive, the performance optimizations already discussed result in processing data in a timely fashion.

B. Visualization

All tests were done on a laptop computer with an Intel Core i7 Q820 processor, the NVIDIA GeForce 280M GTX graphics card with 1GB of GPU memory, 4GB of system memory, Intel X25-M G2 SSD and the Windows 7 OS. The images were rendered at resolution 768 x 512. Node and Brick pools on the GPU were allocated as 64 MB (256 x 256 x 128 x 8B) and 512 MB (1024 x 1024 x 512 x 1B) 3D textures respectively.

The visualization was tested on two data-sets: (1) Hurricane Ike occurrence at 7:10 (GMT), 9/13/2008, (2) tornado outbreak caused by a supercell at 3:20 (GMT), 04/25/2010; by zooming in from the view of the entire US to a close up view of a region where severe weather took place and rotating around in 360 degrees. The regions of interest were Louisiana state for the Hurricane Ike dataset and Kentucky state for the tornado dataset. The images depicting the severe weather regions are shown in Figures 7 and 8, respectively. Rendering times are summarized in Table III.

As expected, the main bottleneck of the visualization is due to loading bricks from the hard drive. To improve the interactivity of the visualization during the camera manipulation, the rendering switches to low resolution setting by decreasing image resolution four times in width and height reaching up to 20 FPS. Please see also the accompanying video at http://hpcg.purdue.edu/bbenes/NEXRAD.mov.

	Average	Maximum
One-pass Ray casting	$0.06s$	$0.12s$
Brick decompression	$0.6s$	$13s$
Node & Brick pool copy to GPU	$0.02s$	$0.02s$
Low resolution visualization	$0.07s$	$0.5s$
High resolution visualization	$1.2s$	$14s$

Table III
TIMING OF EACH STEP DURING VISUALIZATION.

In addition to volume visualization, the high-resolution 2D reflectivity visualization is still possible using maximum intensity projection (MIP) over the generated data structure. Figure 8 shows the comparison with the NOAA's 2D close-up visualization. MIP visualization on the left helps to easily spot regions of high reflectivities. Not only the MIP has much higher resolution, but apparently in the full 3D visualization we can always rotate the view and zoom in to see more details. Figure 9 shows additional results using the MIP visualization.

We also encourage the reader to see the accompanying video for more results.

VII. Conclusions and Future Work

We have presented a system for large-scale volumetric visualization of streaming reflectivity data from multiple

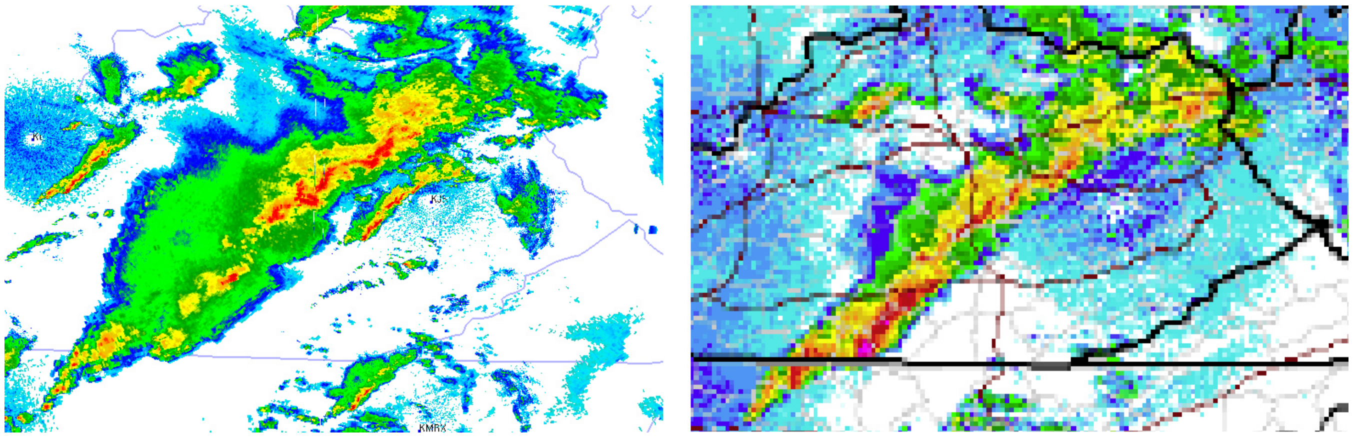


Figure 8. Comparison of our maximum intensity projection visualization (left) with NWS's 2D visualization of all radar data combined in its highest resolution visualization. The slight difference in the images is for the NWS data to be at timestamp of 03:18 GMT, while our data is from 03:20 GMT of 4/25/2010 and also the NWS data has gone through removal of data clutter.

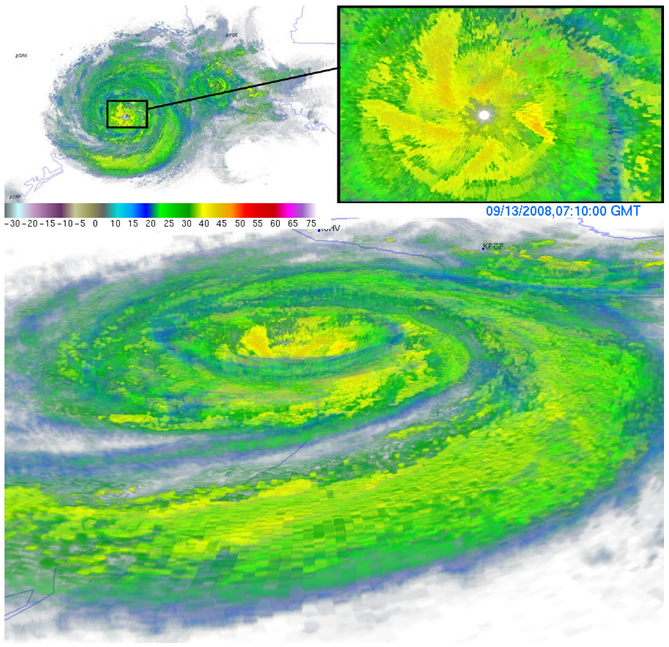


Figure 7. The large-scale 3D visualization of Hurricane Ike over Galveston on 9/13/2008.

Doppler radars. We achieved our main goal of preprocessing the data in a way that promotes effective and high quality large-scale volumetric visualization with required run-time data processing. Although the implementation is not yet fully optimized, it already provides acceptable interactivity while dealing with large streaming datasets.

However, there are still several ways to improve the visualization performance and quality. The rendering algorithm should provide a fall back to a lower resolution brick that is available in the working set on GPU while the higher resolution brick is loaded. The contribution from bricks along the ray have to be filtered to provide smooth transitions between different resolution levels or noticeable artifacts occur when there is a change of resolution. The transfer function provides a good way to filter put unimportant data but does not entirely eliminate all visual clutter. This could be alleviated by adding volume clipping feature such as plane or box. Using the resolution samples and interpolation within each radar's coordinates we managed to capture even the highest frequencies in the data set. The data resolution and coverage could be further improved by including data from TDWR radars at airports near major cities in the U.S. Moreover, our quadtree does not conform to a round earth such as [12]. The lossless RLE compression scheme could be substituted with a lossy 3D wavelet compression scheme [13] to allow for even higher compression ratios while still retaining high frequency volume data.

The weather data have large temporal coherence, and thus it would be beneficial to locally update only the areas that have changed. In addition, current spatial partitioning structure could be extended and optimized to contain data for multiple time events at the node/brick level ([14], [15]) to allow for an interactive temporal visual exploration of weather events.

From our informal study with researchers from Weather Sciences we observed that the users were enthusiastic about the capability of visualizing the global features that span over more than one radar. They mentioned several features that would be useful, such as the ability to follow certain atmospheric phenomena such as super cells. Also, different visualization modes, such as isosurfaces or streamlines would be useful. Last, to provide even better tool for weather analysis, the visualization should combine and display other types of data, such as wind velocity, spectrum width, temperature field and warnings.

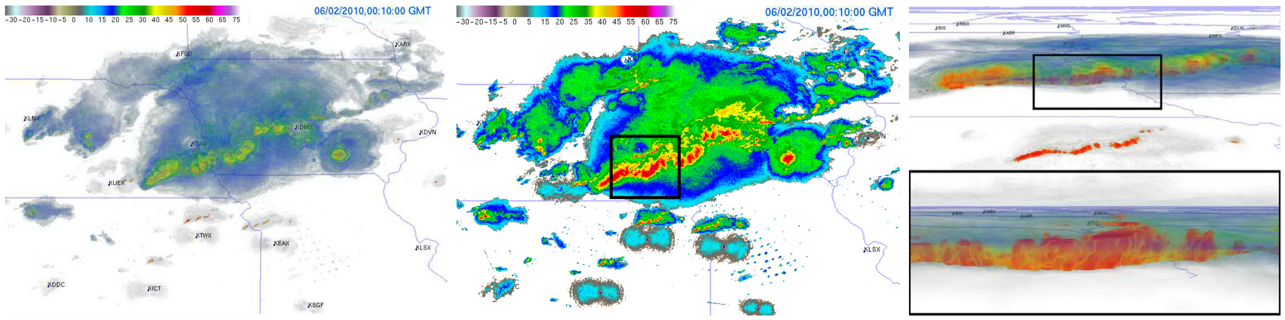


Figure 9. Visualization of reflectivity data using our system. This is visualized using volumetric rendering (left) and MIP visualization (center). MIP visualization allows to spot regions of high reflectivity more easily. After that a user can switch back to volumetric rendering, zoom in and rotate the camera (right) to analyze the vertical domain.

Acknowledgments

We would like to thank Jeff Trapp and his graduate students for providing comments and suggestions for our Doppler reflectivity visualization system. Our thanks also go to David Braun for insightful discussions on how to execute preprocessing on the TeraGrid machines. We would also like to thank to Jin Ryong Kim for his help with the work on Google gadget utilizing our system.

References

[1] M. Huber and J. Trapp, "A review of nexrad level ii: Data, distribution, and applications," *Journal of Terrestrial Observation*, 2005.

[2] Y. Xiao, L. Liu, and Y. Shi, "Study of methods for three-dimensional multiple-radar reflectivity mosaics," *SCI Meteorological Journal*, 2008.

[3] J. Jang, W. Ribarsky, C. D. Sha, and N. Faust, "View-dependent multiresolution splatting of non-uniform data," in *VISSYM '02: Proceedings of the symposium on Data Visualisation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, pp. 125–ff.

[4] L. Jay Miller, C. G. Mohr, and A. J. Weinheimer, "The simple rectification to cartesian space of folded radial velocities from doppler radar sampling," *Journal of Atmospheric and Oceanic Technology*, vol. 3, no. 1, pp. 162–174, 1986.

[5] J. Zhang, K. Howard, and J. J. Gourley, "Constructing three-dimensional multiple-radar reflectivity mosaics: Examples of convective storms and stratiform rain echoes," *Journal of Atmospheric and Oceanic Technology*, vol. 22, 2005.

[6] V. Sundaram, Y. Ru, B. Beneš, L. Zhao, X. C. Song, T. Park, G. Bertoline, and M. Huber, "An integrated system for near real-time 3d visualization of nexrad level ii data using teragrid." in *TeraGrid 2008 Conference*, 2008.

[7] W. Ribarsky, N. Faust, Z. Wartell, C. Shaw, and J. Jang, "Visual query of time-dependent 3d weather in a global geospatial environment," in *Mining Spatio-Temporal Information Systems*, ser. The Springer International Series in Engineering and Computer Science, R. Ladner, K. Shaw, and M. Abdelguerfi, Eds. Springer US, 2002, vol. 699, pp. 83–104.

[8] M. Hadwiger, J. M. Kniss, C. Rezk-salama, D. Weiskopf, and K. Engel, *Real-time Volume Graphics*. Natick, MA, USA: A. K. Peters, Ltd., 2006.

[9] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann, "Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering," in *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, ACM. Boston, MA, Etats-Unis: ACM Press, feb 2009.

[10] R. Fraedrich, J. Schneider, and R. Westermann, "Exploring the millennium run - scalable rendering of large-scale cosmological datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1251–1258, Nov. 2009.

[11] M. Hadwiger, J. Beyer, W.-K. Jeong, and H. Pfister, "Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 12, pp. 2285–2294, 2012.

[12] Z. Wartell, E. Houtgast, O. Pfeiffer, C. Shaw, W. Ribarsky, and F. Post, "Interaction volume management in a multi-scale virtual environment," in *Advances in Information and Intelligent Systems*, ser. Studies in Computational Intelligence, Z. Ras and W. Ribarsky, Eds. Springer Berlin Heidelberg, 2009, vol. 251, pp. 327–349.

[13] I. Ihm and S. Park, "Wavelet-based 3d compression scheme for very large volume data," in *In Proceedings of Graphics Interface '98*, 1998, pp. 107–116.

[14] H.-W. Shen, L.-J. Chiang, and K.-L. Ma, "A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree," in *Proceedings of the conference on Visualization '99: celebrating ten years*, ser. VIS '99. Los Alamitos, CA, USA: IEEE Computer Society Press, 1999, pp. 371–377.

[15] Z. Du, Y.-J. Chiang, and H.-W. Shen, "Out-of-core volume rendering for time-varying fields using a space-partitioning time (spt) tree," in *Proceedings of the 2009 IEEE Pacific Visualization Symposium*, ser. PACIFICVIS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 73–80.