We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,300

Open access books available

172,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

A New Object Selecting Technique for Visualization of Large Datasets

Navid Fallah and Ali Eydgahi

University of Maryland Eastern Shore

Department of Engineering and Aviation Sciences

Princess Anne, MD 21853

aeidgahi@umes.edu

Abstract

This paper presents a new method for accelerating volume rendering of large datasets similar to urban areas. The proposed method improves the current algorithms by introducing a new technique which uses statistics about different objects to cover more general and comprehensive datasets. In this method, proximity, level of details, and statistics from previous moves are used to predict next position and to select objects for future use. This technique keeps track of previous moves and uses the application and viewer's specifications to produce probability areas. These areas help the method to predict next moves and the objects that have to be selected for future use. This technique simplifies the implementation and requires smaller memory footprint. A multi-resolution approach is introduced which is based on a special shape. By gathering statistics about different objects a unique shape is obtained that is used to select different objects and put them in the groups to be rendered with different resolutions. This way acceleration to produce high resolution visualization at interactive rate is achieved. The proposed technique solves the issues of redundant object downloading and redundant rendering on the server by downloading objects and images based on software and hardware specifications. This way it saves server and network resources to reduce the delay and to achieve interactive frame rate on client side. A visualization program based on the proposed method is presented.

KEY WORDS

Visualization, volume rendering, object selecting, simulation, multi-resolution, and large datasets.

1. Introduction

An increasing number of urban and geographic applications are now generating high resolution three-dimensional (3D) datasets. The sizes of these datasets are often too large to fit into memory or even on hard drive. This makes it almost impossible to keep the data on

hard drive and load them into memory to perform interactive data visualization. One example is the Richtmyer-Meshkov Turbulent simulation [1], which is designed to study instabilities at the interface between two gases of different densities and produces datasets containing hundreds of time steps each being 7.5 gigabytes in size.

In the past few years, research has suggested that a considerable portion of a large dataset is often invisible due to the view angle and various aspects of data coherence could be exploited to reduce the amount of data that pass through the visualization pipeline [2]. An effective technique for reducing unnecessary rendering computation by eliminating invisible portions of data before visualization is presented as occlusion culling [3]. Guthe and Strasser [4] applied visibility test to multi-resolution volume rendering, Livnat and Hansen [5] introduced a view-dependent algorithm for iso-surface extraction, and several predictive approaches for structured datasets have been presented in [6]-[8]. Increasing gap between the available I/O, memory and computing bandwidth, and the rapidly increasing amount of data to be visualized requires new methods to select precisely the right objects to be loaded into memory.

Most of the desktop machines or laptops are not able to hold the large dataset due to their limited size of storage and memory. Therefore, techniques based on client-server architecture are used to solve this problem. One of the methods is unstructured grids introduced in [9]. The visibility sorting algorithm [10] is another technique, which sorts in object-space and image-space. This algorithm is fast, efficient, and flexible enough to handle geometry that are dynamically changing [11]. The hardware assisted progressive volume rendering introduced in [12] creates a progressive image transmission over the internet. In this method, a server renders objects and the produced images are sent to the client to be shown on screen. In this case, only a few images are stored on the client computers [12]. Having just the images on the client but not the objects slows down the exploration of interactive dataset. For example, when the view direction changes just a few degrees, the same objects with different angle have to be shown on screen and new images have to be rendered on server side and be sent to client. This uses both the server resources and network bandwidth and introduces new delays.

Direct volume rendering has become a standard technique for visualizing 3D datasets. This technique maps the data in the volume and projects it to 2D images. The direct volume rendering techniques using 3D texture mapping and hardware can visualize volumes of moderate sizes at interactive frame rates. The challenge is to allow interactive data exploration for even larger datasets. While the available texture memory in the high-end graphics hardware is limited to only several hundred megabytes, nowadays simulation applications can produce terabytes of data [13]. A practical solution for this issue is to reduce the amount of data being rendered. A possible technique [14] is to give the user a quick overview of the data. It is useful to first render the data at a lower resolution. As the user navigates through the data and requests further details in local regions of interest, if the rendering resources are available, different portions of the data are then retrieved and rendered at their higher resolution.

As visualization is an interactive process, sometimes rendering a lower resolution of data is sufficient for the user to get enough information about the area based on its distance. The capability to visualize data at different resolutions allows user to focus on specific area of interest and to spend more time and hardware resources for higher quality data for that area. Techniques have been introduced to provide hierarchical data representations for 3D volumetric data in [15]-[17] and for multi-resolution encoding and rendering of large scale in [18].

Level of detail (LOD) based visualization techniques as proposed in [19] allow rendering of the same objects using several different triangle meshes of variable complexity. Thus, the mesh complexity can be adjusted according to the object's relative position from the viewer, its visual importance in the rendered scene, and with respect to the available rendering power to guarantee stable interactive frame rates [20]. Although numerous ideas have been implemented on mesh simplification for geometric approximation [21]-[22], fewer approaches have been taken to address the problem of view-dependent simplification [20] for real-time rendering and performance optimization.

The proposed method in this paper, improves these methods by introducing new technique to predict and cover more general and comprehensive datasets. The method keeps track of previous moves and uses the application and viewers' specifications to produce probably areas. These areas help the method to predict next move and the objects that have to be selected for future use. The proposed method introduces a LOD technique which has been optimized for urban area visualization and requires minimum amount of necessary memory. This method is based on three layers of details. The first layer can be of different shape depending on the rendering objects. The second layer selects the objects for future use based on an algorithm specialized for this method and cache them into memory. The last layer downloads the objects over the network based on the same algorithm. The advantage of these layers of details is in their specific shape which is illustrated in the next sections.

2. The Proposed Method and its Data Structure

In the proposed method, different concepts have been employed to accelerate algorithms for visualizing large datasets. These include a customized data structure, a probability area based on different aspects of movement, different algorithm phases for organizing the dataset and selecting objects to visualize, and extending the algorithm to 3D and using it in network. A particular data structure to organize datasets and minimize the access time to objects is introduced. This data structure is optimized to have minimal overhead and best performance at different phases such as adding new objects to an existing dataset.

The data structure consists of three different types of objects. A prefixed is added to each type to distinguish between these objects. The first objects are Geo-Objects which have to be shown on screen. Geo-Objects simulate the real word objects and keep different information about each object such as shape, color, and location. The two other types of objects are Container-Objects and Linker-Objects. These two types construct the platform for data structure by pointing to each other and to relevant Geo-Objects. These objects are responsible to keep the integrity of the Geo-Objects.

Each Container-Object refers to specific geographic area and points to a group of Geo-Objects belonging to the same geographic location. These objects have to attach to each other in order to construct the dataset using Container-Objects. Each Container-Object by using Linker-Objects connects to all other Container-Objects around it. Container-Objects have pointer to point to all necessary Linker-Objects that are around it. Linker-Objects are pointing to Container-Objects that are next to them and other Linker-Objects. Connecting Container-Objects using this technique helps the algorithm to find nearby Geo-Objects in least amount of time.

Container-Objects have size-limit based on hardware and software specifications and requirements. By adding each Geo-Object to a specific location, the relevant Container-Object for that area points to that Geo-Object. When the size of the Geo-Objects that are being pointed with Container-Object exceeds the size-limit for that Container-Object, the Container-Object has to be broken down to smaller pieces and cover the same area with more Container-Objects to keep the balance. This feature is necessary to improve the memory performance and keep the access time as short as possible.

As a Container-Object breaks down, new Linker-Objects joins the new Container-Objects and other Container-Objects that are nearby. Separating these two responsibilities, maintain Geo-Objects with Container-Objects and establishes the links between these objects, which improves the memory and CPU performance while updating and maintaining the objects.

Fig. 1. Different types of objects used in the proposed data structure.

Figure 1 show the three different objects used in this data structure. The white objects represent Geo-Objects which will be rendered and will be shown on screen at the same time. Dark gray represent Container-Objects which points to Geo-Objects in their area. When a special area needs to be shown on screen, that container and the related Geo-Objects are loaded into memory to render. The bright gray represents Linker-Objects which links different containers. These objects are being used to find neighbors of a container when they need to be selected.

Handling the data structure consists of two phases. The startup phase and update phase which includes add or delete of Geo-Objects to/from an existing dataset.

The startup phase starts with an empty Container-Object which covers all the demanded area. Geo-Objects are being added to the container till the container reaches the size-limit. The size-limit of the container depends on the software and hardware specifications such as the rendering device, graphic card, and main memory. When the container reaches the size-limit, it is broken down to four new containers which each covers one-fourth of the old container area. This process continues until all the Geo-Objects have been added to the containers. Since each break produces 4 new containers, frequently breaking is not necessary and it improves the performance. Figure 2 demonstrates process of breaking a container into four new containers when size-limit is reached.

Fig. 2. The process of breaking a container which has reached the size-limit.

At the first phase while importing the dataset, the frequency of breaking the containers is high. This is only in the startup phase. Later changes on dataset need less effort and time. Thus, this method helps to speed up the access time and to retrieve data in less amount of time.

For the update phase, there are some Geo-Objects that need to be added or deleted. Process of adding Geo-Objects in this phase is the same process as adding Geo-Objects at the startup phase. Again, container-Objects are being broken to four new ones when they exceed the size-limit. By deleting Geo-Objects, the size of containers reduces. The containers that have

less than one-eighth of size-limit have to be combined with their neighbors to make a new container in order to avoid wasting storage and memory spaces. If all four containers have one-eighth of size-limit, by combining them the new container will have half of the size-limit.

One of the advantages of this data structure is not being dependent on the shape and distribution of objects. The data structure does not waste storage by assigning extra space based on specific geographic distribution over the area. For example, a diamond-shaped distribution of objects seems to have wasted half of the containers as shown in figure 3. By using the proposed data structure, the memory is allocated just for the parts that contain Geo-Objects. Figure 4 demonstrates the same dataset representation using the proposed method. When a container is empty there is no need to allocate any storage and memory for the relevant Container-Object. Linker-Objects have to keep track of links and in this case they point to other linkers and as there is no container the pointers will be null.

Fig. 3. Representation of a sample diamond dataset.

To speed up access time to objects, containers have to be numbered in specific order. This method of numbering helps to find any Container-Object in the area and its related containers such as its neighbors or children's of the same container in case the containers have to be merged. The merge is necessary when the four containers from the same parents have less than one-eighth of size-limit. An example of numbering process after a container is broken to new containers is shown in figure 5.

Fig. 4. Representation of a sample diamond dataset using proposed data structure.

For the first break, numbering process is as shown in figure 5. In this process, containers are numbered using two digits where the first digit is the number of main container. For example, by breaking container number 2 to four new containers, they are numbered as 20, 21, 22, and 23. This number specifies the relevant container in reverse method. For example, having container numbered 321, 3 means in the first break this container was a part of a container in South-East of the data set, 2 means that in second break this object was part of the container in South-West of the old 3 container, and finally, 1 means in the last break this container is in North-East of the old 32 container. By using this numbering method, any container or Geo-Object can be identified very fast and finding the neighbors of a container is achieved just by changing a digit in this string of digits.

Fig. 5. Numbering a container after it is broken to four new containers.

The other advantage of this numbering is that each digit can be represented only by 2 bits as L_nD_n . For example, container numbered 32012203 can be stored in 16 bits. Bitwise instructions can be used to manipulate these numbers and to extract container information such as location and neighbors which reduce the calculation time. Figure 6 demonstrates a sample container that has been broken many times and its corresponding container numbering.

The algorithm for finding neighbors of a container is recursive and uses bitwise operations to locate them. As is shown in figure 5, the first bit for the left containers is always 0 and for right containers is 1. For the containers on top half of the second bit is 0 and for the down half it is 1. The first step in the process of finding the neighbors in left or right is to change the first bit. For example, to find 102 or 010010's neighbors, by changing the first bit the container 010011 or 103 is selected. To find the neighbor below or above, the second bit has to be changed which in this case it is 010000 or 100.

For the next step in the process of finding the neighbors, container numbers are represented as $(L_n D_n \dots L_1 D_1 L_0 D_0)$ where L_i and D_i are the two bits of a digit. To find neighbor of a container that is located above or below, we consider:

$$L_0 = \overline{L_0} ,$$

$$L_1 = \overline{L_1} ,$$

$$L_i = (\overline{L_{i-1} \oplus L_{i-2}}) \oplus L_i \quad \text{for } i > 1$$

Where bar symbol indicates logical not operation. This way, we find a neighbor in either above or below the container and then by changing the first bit, D_0 , from 0 to 1 or from 1 to 0 the other neighbor is obtained.

To find the containers on left or right of an object, we consider:

$$D_0 = \overline{D_0} ,$$

$$D_1 = \overline{D_1} ,$$

$$D_i = (\overline{D_{i-1} \oplus D_{i-2}}) \oplus D_i \quad \text{for } i > 1$$

This way, we find a neighbor in either left or right of the container and then by changing the second bit, L_0 , from 0 to 1 or from 1 to 0 the other neighbor is obtained.

As an example, we consider 102 or 010010 container. Thus, L_0 is 1, L_1 is 0, L_2 is 0, D_0 is 0, D_1 is 0, and D_2 is 1. For the containers above and below, the L_0 will be 0, the L_1 will be 1, and the L_2 will be 0. Two new neighbors are 011000 or 120 and 011001 or 121. Left and right neighbors can be found with the same method.

Fig. 6. Sample dataset demonstrating container numbering.

3. The Layers

In this paper, a specific shape called layer is introduced. The layer is obtained from probability of viewers changing their directions. The layer is used to identify and select proper objects that need to be processed. Different statistics and characteristics of movement such as speed and probability of turning to different directions are employed to generate layers. The layer is used to predict the viewer’s next position and consequently enables the algorithm to deal with the objects that are affected by the move. This improves the performance, reduces the delay time, and makes it feasible for the real-time visualization of objects.

It is known that the probability of viewer moving forward is the highest, turning back is the lowest, and turning left or right is in between. Figure 7 shows a sample of layer.

Fig. 7. A sample layer.

In figure 7, VP is the view point and F, B, R, and L represent the probability of going straight, turning back, turning right, and turning left. This is the best shape for layers and can be used for any kind of viewer. Viewer can be a car, airplane, human, or any type of object which simulate the viewer movement. The only difference between layers of different viewers is in their probability for turning. The shape is the same for all viewers and the algorithm is the same. The layer has to be customized and defined for each type of viewer.

There are two methods to generate the layers. The first one is to provide the probabilities while defining a new object as viewer. The second approach is to use the same probability for all directions as default. In this case, the program keeps track of movements and collects necessary statistics to generate probabilities and make corrections on the layers shape.

Since human moves slower than the other two viewers, it takes more time to get to farthest points and it is more likely for it to turn left or right than other two viewers. Therefore, it is necessary to have more details about nearer Geo-Objects and to select more Geo-Objects in left or right than other viewers. The airplane moves faster than the other two viewers and gets to farther points in less time. Therefore, selecting far Geo-Objects are inevitable and since it needs less detail in its left or right fewer Geo-Objects is selected in these directions.

The distance between the viewer and Geo-Objects around it creates layers with different sizes. Combining layers with different sizes produce a base layer which is used by the algorithm to determine proper resolution for each Geo-Object using multi-resolution and LOD concepts. Figure 8 shows a base layer created by four layers of different sizes.

Fig. 8. A sample base layer.

The interior layer selects the Geo-Objects that have to be shown on screen and have to be rendered by an algorithm. Other layers select the Geo-Objects that do not need to be rendered but have to be cached to memory for later use. This technique speeds up the rendering process for next move when interior layer covers the Geo-Objects. The interior layer can contain several sub-layers depending on specification of an environment. The algorithm can render Geo-Objects in each sub-layer with different resolution and spend more time for Geo-Objects that are closer to viewer. Using this special shape helps to render or cache Geo-Objects into the memory more accurately by reducing the effort for caching the Geo-Objects that are less likely to be visited by viewer.

4. The Algorithm

The algorithm uses the probability area to select proper objects and applies the related future. Each layer is responsible to select the relevant objects for specific process. The size and number of layers depend on hardware and software specifications. Depending on the layer, the corresponding objects are selected either to render with proper resolution, to be cached into memory, or to be downloaded from the server.

This saves time by eliminating the delay time for downloading objects at rendering phase. Accurate prediction of required objects for future use is essential for reducing delay and achieving better interactive rate for visualization. The proposed layer is designed to achieve the better interactive rate.

The three rendering, caching, and downloading phases are parallel processes. These three processes use graphic hardware, memory, and network resources that are three different resources that do not interfere with each other and can work as a pipeline.

The algorithm selects the objects by moving the layer over the dataset. The objects in each layer are identified by moving the area based on the viewer speed and direction. This method employs queues for each layer to keep track of objects that have to be selected.

Selection process starts with the container which includes the viewer and pushes the container to the first layer's queue. In each step, first item in queue will be popped up and its neighbors will be selected and pushed to the proper queue. If they are in the same resolution area, they will be added to current queue. Otherwise, depending on their layer, they will be pushed to other queues either for lower resolution areas to be cached into memory or to be downloaded from the server. This will continue till all neighbor objects are selected and are placed in a proper area. The next step is to process the objects which includes rendering with proper resolution, caching into memory, or downloading.

The images from previous renders that are stored on hard drive or downloaded from the server are used to reduce the delay time. These images can be shown on screen while the algorithm works on the queuing or rendering process. The new images can be replaced on screen when the rendering is finished. These images can be downloaded from the server, or from previous rendering process that have been stored, or can be rendered while hardware resources are not in use. This can be done for containers that do not have images or the ones that have changed from the last time that they have been rendered.

5. An Example

A sample area is applied on sample dataset as is presented in figure 6 to demonstrate the object selection process. Figure 9 shows the sample probability area with two layers.

Fig. 9. Sample probability areas with two layers.

Figure 10 shows the result of applying the probability area in figure 9 to sample dataset in figure 6. The viewer is in north east of 023 container.

Fig. 10. Selected containers with the applied sample layers.

The algorithm starts with the container that includes the viewer. The algorithm pushes the container into the first queue, which is the rendering layer. The objects in this layer have to be rendered. The algorithm finds the neighbors of 023 container by modifying the digits in container's number. For the 023 container, neighbors are 020, 021, 022, 311, 3100, 3101, 30, 032, and 031. As 021 and 022 containers are in the same layer, they are pushed into the first queue. 020 and 311 containers are in next layer, and they are placed in the second queue. The rest of containers are not in probability areas and they are not placed in any layer and

queue. For the next step as the 021 container is on the top of first queue, 1303 is pushed into the first queue and 1300 is pushed into the second one. For 022 neighbors, 1330 and 1333 are pushed into the first queue and 200 will be pushed into the second queue. For the next step, 1303, 1331, 1332, 131, and 132 are placed into the first queue and 1301, 201, 120, 1230, 1233, 2100, 1231, 1232, 1213, and 122 are placed into the second queue. The rest of containers are not in any layer. Selecting phase is complete as the area of each object is determined.

6. Implemented software

A program in java has been implemented to demonstrate the effectiveness of the proposed algorithm. The program consists of RLMain, RLFrame, DataSetCreator, DataSetViewer, RLImageApplet, and RLColorApplet classes. In this software the rendered images are shown on screen and rendering phase can be added as an extension.

The DataSetCreator class creates the dataset based on the proposed method and uses images as its input. This class puts the images in proper container and keeps images with three different resolutions.

The DataSetViewer class shows the dataset made by DataSetCreator class. This class shows images in the dataset with random resolution for sample and quick demonstration of the dataset produced by DataSetCreator. It also uses other classes to implement other functionality of the proposed method.

The RLImageApplet class uses the dataset produced by DataSetCreator class to implement the selecting phase of the proposed method. This class is a Java applet and selects containers, puts them in different resolution areas, and displays them on screen based on their proper resolution. There are three different resolution areas in the algorithm. A rectangle simulates the screen position on dataset and the area surrounded with the rectangle to clearly demonstrate the method. The probability area has two layers. The first layer is internal layer which selects the objects that are close to viewer. The objects in this area have to be rendered with the best resolution. The second layer selects the objects to be rendered with lower resolution as they are far from the viewer. These layers also select extra objects which are not in screen area and user is not able to see them. These extra objects are cached for future use to reduce the delay for later moves. Depending on the size of probability areas, some objects that have to be shown on screen may not be selected by any layer. In this case, the objects have to be selected as a new area and to be rendered with lower resolution.

The RLColorApplet class uses three different colors to simulate the proposed method. The screen simulation and probability areas are the same as RLImageApplet class. In this class to simplify the demonstration, each color represents a specific resolution area. The three colors and their corresponding layers are:

- For the layer with the highest resolution.
- For the layer with intermediate resolution.
- For the layer with the lowest resolution.

Figure 11 shows the selected areas with RLImageApplet and RLColorApplet. The objects enclosed within the internal layer are rendered with the highest resolution. The objects in second area are displayed with lower resolution. Lowest resolution applies to the objects that are in screen rectangle but not selected by any probability area. The images in figure 11 represent the selected areas chosen by the algorithm before viewer moves.

Fig. 11. Selected areas before viewer moves.

Figure 12 shows the selected areas after forward move. Since most of the necessary objects had been selected and rendered for previous move, the images in this figure show there are only few objects that need to be rendered and to be shown on screen for new position of the viewer.

Fig. 12. Selected areas after forward move.

Figure 13 shows the selected areas after turning to the right. As probability areas represent, there are less objects ready on the sides other than front side of the viewer. In case of turning to the sides, more objects have to be selected and to be rendered with the proper resolution because it is a slower process than moving forward. Therefore, since the probability of turning to the sides is less than probability of moving forward, it provides minimum delay.

IntechOpen

Fig. 13. Selected areas after right turn.

7. Conclusion

In this paper, a new method has been introduced to improve visualization of large datasets performance. A program based on the new method which utilizes proximity, level of details, and statistics from previous moves to predict next position and to select necessary objects for future use has been presented. This program is not dependent on the shape and distribution of objects and does not waste storage by assigning extra space based on specific geographic distribution over an area.

One of the advantages of the proposed algorithm is when datasets are too large to fit into local hard drive or when the geo-objects are frequently updated. If the available hard drive is large, more geo-objects can be kept on client side. But, if available storage size is small, geo-objects have to be downloaded more frequently. If geo-objects change repeatedly, as it is the case most of the times, the updated geo-objects have to be downloaded. Thus, it is not efficient to keep many geo-objects on hard drive. If the geo-objects are not changed frequently, objects can be stored on hard drive depending on storage capacity. This reduces the need for downloading geo-objects frequently and improves the performance.

The best shape for the probability area has been discussed to ensure the algorithm's interactive navigation. A set of effective view-dependent strategies has been suggested for selecting objects with small memory footprint and efficient use of network and server resource. Level of details, multi-resolution, and pipeline are the methods incorporated into the proposed technique to improve its performance.

8. Acknowledgment

This work was supported by NASA Langley Research Center Grant # NOC1-03033 under the Chesapeake Information Based Aeronautic Consortium (CIBAC).

9. Reference

- A. A. Mirin, R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimits, M. A. Duchaineau, D. E. Eliason, D. R. Schikore, S. E. Anderson, D. H. Porter, P. R. Woodward, L. J. Shieh, and S. W. White, "Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System," Proc. of the 1999 ACM/IEEE Conference on Supercomputing, Portland, Oregon, pp.70, 1999.
- J. Gao, H. Shen, J. Huang, and James Kohl, "Visibility Culling for Time-Varying Volume Rendering Using Temporal Occlusion Coherence," Proc. of IEEE Visualization Conference, Austin, TX, pp. 147-154, 2004.
- N. Govindaraju, A. Sud, S. Yoon, and D. Manocha, "Parallel Occlusion Culling for Interactive Walkthroughs using Multiple GPUs," University of North Carolina, Charlotte, Computer Science Technical Report TR02-027, 2002.
- S. Guthe and W. Strasser, "Advanced Techniques for High-Quality Multi-Resolution Volume Rendering," Computer & Graphics, vol. 28, no. 1, pp. 51-58, 2004.
- Z. Liu, A. Finkelstein, K. Li, "Improving Progressive View-Dependent Isosurface Propagation," Computers & Graphics, vol. 26, no. 2, pp. 209-218, 2002.

- L. Ibarria, P. Lindstrom, and J. Rossignac, "Spectral Predictors," IEEE Data Compression Conference, Snowbird, UT, 2007. To appear.
- B. H. Faria, "Linear Predictive Transform of Monochrome Images," Image and Vision Computing, vol. 5, no. 4, pp. 267-278, 1987.
- L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-Core Compression and Decompression of Large n-Dimensional Scalar Fields," Proc. of Eurographics, Granada, Spain, pp. , 2003.
- C. T. Silva, J. L. D. Comba, S. P. Callahan, and F.F. Bernardon, "A survey of GPU-Based Volume Rendering of Unstructured Grids," Brazilian Journal of Theoretic and Applied Computing (RITA), vol. 12, no. 2, pp. 9-29, 2005.
- S. P. Callahan, M. Ikits, J. L. Comba, and C. T. Silva, "Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering," IEEE Transactions on Visualization and Computer Graphics, vol. 11, no. 3, pp. 285-295, 2005.
- F. F. Bernardon, S. P. Callahan, J. L. D. Comba, and C. T. Silva, "Interactive Volume Rendering of Unstructured Grids with Time-Varying Scalar Fields," Proc. of Eurographics Symposium on Parallel Graphics and Visualization, Lisbon, Portugal, pp. 51-58, 2006.
- S. P. Callahan, L. Bavoil, V. Pascucci, and C. T. Silva, "Progressive Volume Rendering of Large Unstructured Grids," IEEE Transactions on Visualization and Computer Graphics, vol. 12, no. 5, pp. 1307-1314, 2006.
- J. Gao, C. Wang, L. Li, and H. Shen, "A Parallel Multiresolution Volume Rendering Algorithm for Large Data Visualization," Parallel Computing, vol. 31, no. 2, pp. 185-204, 2005.
- I. Boada, I. Navazo, and R. Scopigno, "Multi Resolution Volume Visualization with a Texture-Based Octree," The Visual Computer, vol. 17, no. 3, pp. 185-197, 2001.
- D. Ellsworth, L. J. Chhian, and H. W. Shen, "Accelerating Time-Varying Hardware Volume Rendering Using TSP Trees and Color-Based Error Metrics," Proc. of the 2000 IEEE symposium on Volume visualization, Salt Lake City, UT, pp. 119-128, 2000.
- S. Guthe, M. Wand, J. Gonser, W. Strasser, "Interactive Rendering of Large Volume Data Sets," Proc. of IEEE visualization, Boston, MA, pp. 53-60, 2002.
- T. Funkhouser and C. Sequin, "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environment," Proc. of the 20th Annual Conference on Computer Graphics and Interactive Techniques, Anaheim, CA, pp. 247-254, 1993.
- R. Pajarola and C. DeCoro, "Efficient Implementation of Real-Time View-Dependent Multiresolution Meshing," IEEE Transactions on Visualization and Computer Graphics, vol. 10, no. 3, pp. 353-368, 2004.
- P. Cignoni, C. Montani, and R. Scopigno, "A Comparison of Mesh Simplification Algorithms," Computer & Graphics, vol. 22, no. 1, pp. 37-54, 1998.
- P. Lindstrom and G. Turk, "Evaluation of Memory-less Simplification," IEEE Trans. Visualization and Computer Graphics, vol. 5, no. 2, pp. 98-115, 1999.
- C. Wang and H. Shen, "Hierarchical Navigation Interface: Leveraging Multiple Coordinated Views for Level-of-Detail Multiresolution Volume Rendering of Large Scientific Datasets," Proc. of Ninth International Conference on Information Visualization, Greenwich, UK, pp. 259- 267, 2005.
- B. Shneiderman, "The Eyes Have it, A Task by Data Type and Taxonomy for Information Visualizations," Proc. of IEEE Visual Languages, Boulder, CO, pp. 336-343, 1996.

IntechOpen

IntechOpen

Advances in Computer Science and IT

Edited by D M Akbar Hussain

ISBN 978-953-7619-51-0

Hard cover, 420 pages

Publisher InTech

Published online 01, December, 2009

Published in print edition December, 2009

The book presents some very interesting and excellent articles for this divergent title. The 22 chapters presented here cover core topics of computer science such as visualization of large databases, security, ontology, user interface, graphs, object oriented software developments, and on the engineering side filtering, motion dynamics, adaptive fuzzy logic, and hyper static mechanical systems. It also covers topics which are combination of computer science and engineering such as meta computing, future mobiles, colour image analysis, relative representation and recognition, and neural networks. The book will serve a unique purpose through these multi-disciplined topics to share different but interesting views on each of these topics.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Navid Fallah and Ali Eydgahi (2009). A New Object Selecting Technique for Visualization of Large Datasets, Advances in Computer Science and IT, D M Akbar Hussain (Ed.), ISBN: 978-953-7619-51-0, InTech, Available from: <http://www.intechopen.com/books/advances-in-computer-science-and-it/a-new-object-selecting-technique-for-visualization-of-large-datasets>

INTECH
open science | open minds

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2009 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the [Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License](https://creativecommons.org/licenses/by-nc-sa/3.0/), which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen